NUMERO 1 IN ITALIA SU ELETTRONICA EMBEDDED E SEMICONDUTTORI

# Firmware 2.0

#00
Gennaio-Febbraio 2020

# IoT
## (Internet Of Things)

IN QUESTO NUMERO:

- IOT È CHI IOT LO FA
- ESPERTINO E IFTTT UN MAGGIORDOMO TUTTOFARE
- INTEGRARE DISPOSITIVI IOT NELLA RETE IOTA

E MOLTO ALTRO!

it.emcelettronica.com

Elettronica Open Source

PROFESSIONAL & MAKER ZONE

# Firmware 2.0 TOPICS 2020

| QUESTO MESE: | FOCUS DEL MESE | | |
|---|---|---|---|
| IOT | IOT | GEN/FEB | 1 Febbraio |
| CAR HACKING - OBD | AUTOMOTIVE | MARZO | 1 Marzo |
| BLOCKCHAIN | AI | APRILE | 1 Aprile |
| AUDIO/VIDEO | WIRELESS/RF | MAGGIO | 1 Maggio |
| MAKERS SENSOR BOARDS | SENSORS | GIUGNO | 1 Giugno |
| ROBOTICS | POWER/MOTOR | LUGLIO | 1 Luglio |
| SMART PROJECTS | INDUSTRY 4.0 | AGO/SET | 1 Settembre |
| MAKERS LABORATORY | TEST & MEASUREMENTS | OTTOBRE | 1 Ottobre |
| WEARABLE | LED LIGHTING OPTOELECTRONICS | NOVEMBRE | 1 Novembre |
| MAKERS BOARDS | EMBEDDED DESIGN | DICEMBRE | 1 Dicembre |
| MAKERS ZONE | | | |

## COSA LEGGERAI NEL 2020?

# SIAMO LA PIÙ GRANDE COMMUNITY DI ELETTRONICA EMBEDDED E MICROCONTROLLORI!

*La nostra forza non sono i contenuti di qualità che giornalmente i nostri ingegneri realizzano, non sono gli argomenti sempre di attualità che trattiamo, e nemmeno la visione del futuro che abbiamo sempre avuto con successo, anticipando argomenti ormai divenuti popolari come IoT, piuttosto che AI, Blockchain e 5G.*

***No, non è tutto questo che ci sostiene. Il nostro vero valore aggiunto siete voi!***

*La community di Elettronica Open Source, con più di 130.000 utenti registrati sul sito, e più di 100.000 persone che ci seguono sui vari social, oltre a migliaia di abbonati.*

***Tutti noi siamo Elettronica Open Source!***

*E tutti noi stiamo realizzando la migliore rivista di elettronica di sempre che... non è solo una rivista!*
*Infatti Firmware 2.0 è una rivista ma può essere considerata anche un ebook monotematico da conservare e consultare nel tempo.*
*Internet of Things, Automotive, Wireless/RF, Makers Boards, Embedded Design, AI, Blockchain, Power, etc.*
*Questi sono solo alcuni degli argomenti che tratteremo nel corso del 2020 (vedi il piano editoriale completo).*

*Oggi esce il primo numero di Firmware2.0, questo è dedicato all'IoT, con punti di vista professionali e "da makers" sull'argomento. Senza zone distinte, senza separazioni, perché pensiamo che professionisti e makers debbano sempre andare nella stessa direzione! Ovviamente con modalità diverse, il professionista in modo più strutturato e con vari livelli di test, mentre il maker in modalità "learning-by-doing" e spesso più rapido. Ma quello che conta è l'obiettivo, ed è comune: la progettazione elettronica e tutto il mondo che la circonda.*

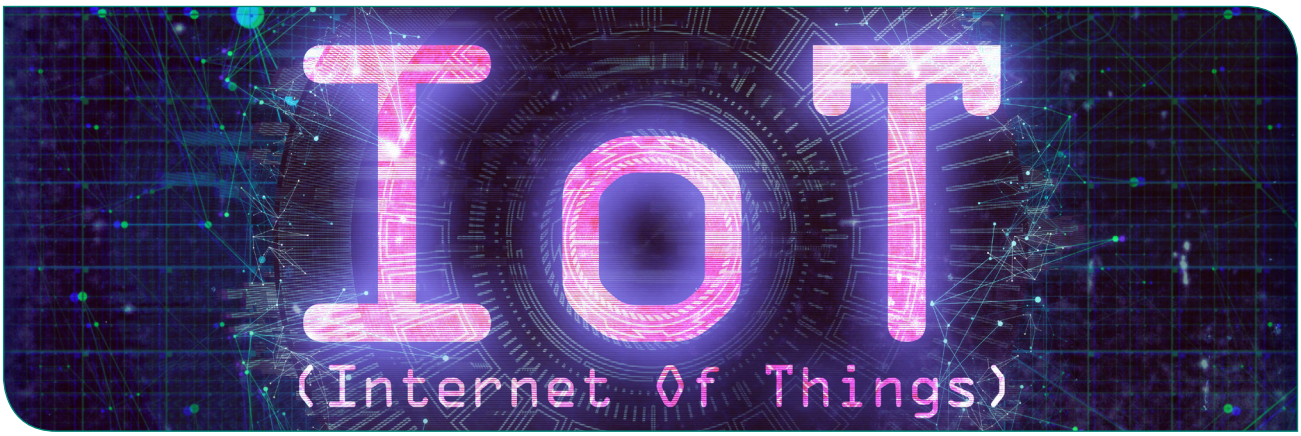
*Grazie a tutti gli abbonati che ci hanno inviato consigli preziosi, a voi dico che la nostra linea editoriale è dinamica, quindi ascolteremo sempre i vostri suggerimenti e ne faremo tesoro.*

***A tutti coloro che ancora non fanno parte di EOS posso solo invitarvi a salire a bordo!***

*Fatevi un giro sul blog, leggete i nostri articoli (o gli estratti) e poi date una svolta alla vostra carriera o al vostro hobby preferito: entrate in Elettronica Open Source e Firmware 2.0*

*Emanuele Bonanni*

# IoT




**Founder&Editor**
Emanuele Bonanni

**CFO**
Lidia Balica

**Editorial Assistant**
Maria Pisani

**Maker in Chief**
Giordana Francesca Brescia

**Marketing**
Sara Ercolani

**Advertising**
Cristian Balica
cristian@contangosl.com

**Graphic Designer**
Marilde Mirra

**Circulation**
Users - 131.586
Social Network - 121.156

**© Copyright**
Tutti i diritti di riproduzione o di traduzione degli articoli pubblicati sono riservati. Manoscritti e disegni sono di proprietà di Contango SL.
E' vietata la riproduzione anche parziale degli articoli salvo espressa autorizzazione scritta dell'editore. I contenuti pubblicitari sono riportati senza responsabilità, a puro titolo informativo.

Autorizzazione alla pubblicazione del Tribunale MI n. 20 del 16/01/2006

**EDITORIALE** Siamo la più grande community di Elettronica Embedded e Microcontrollori! 1

IOT È CHI IOT LO FA: DALL'IDEA AL PROTOTIPO 4

IOT È CHI IOT LO FA: COMPRENDIAMO GLI STRUMENTI 14

IOT È CHI IOT LO FA: IL PORTASPICCIOLI SMART 27

IOT È CHI IOT LO FA: UNO SMART STAND PER IL NOSTRO SMARTPHONE 34

APPLICAZIONI TECNOLOGICHE DI RELATIVITÀ RISTRETTA 39

CONTROLLO REMOTO DI ARDUINO DA DISPOSITIVI MOBILE 44

SEMPLIFICARE PROGETTI IOT BASATI SU STANDARD CELLULARI A BASSA POTENZA 57

AVR-IOT: SCHEDA DI SVILUPPO PER COLLEGARE LE APPLICAZIONI IOT A GOOGLE CLOUD 61

SDK PER ESPERTINO: GUIDA ALL'INSTALLAZIONE 67

ESPERTINO E IFTTT: UN MAGGIORDOMO TUTTOFARE CON IL NOSTRO SMARTPHONE 77

ESPERTINO E IFTTT: PULSANTONE DI EMERGENZA PER ANZIANI 85

PULSANTI PER QUIZ: UN SEMPLICE PROGETTO CON ESPERTINO PER GIOCARE AI CLASSICI GIOCHI DELLA TV 95

APPLICAZIONI CRITTOGRAFICHE DI ESPERTINO 103

CRYPTO TICKER CON ESPERTINO: PROGETTO COMPLETO 111

COME INTEGRARE DISPOSITIVI IOT NELLA RETE IOTA 119

APPLICAZIONI DELLO STANDARD DDS AL MONDO IOT 126

MICROCHIP SEMPLIFICA LA SICUREZZA IOT HARDWARE-BASED CON SOLUZIONI DI PREPROVISIONING 132

DOMUS 1.0 SISTEMA DI CONTROLLO CON GESTIONE REMOTA VIA WEB 136

# Lavori (o vorresti lavorare) nel settore dell'elettronica?

EOS-Academy è l'asso nella manica dei professionisti dell'elettronica che hanno scelto di investire sulle loro potenzialità.

EOS ACADEMY

PER MAGGIORI INFORMAZIONI VISITA LA PAGINA
**https://it.emcelettronica.com/eos-academy**

# IOT È CHI IOT LO FA: DALL'IDEA AL PROTOTIPO

di **Pietro Boccadoro**

*Progettare è un'arte ed è davvero necessario essere un po' artisti per riuscire a farlo correttamente. Quando si crea un progetto, diventa indispensabile immaginare tutto, preventivare ogni singola funzione, ogni comportamento, sia esso dell'utente o del sistema, ed è fondamentale anche specificare ogni variabile di interesse per poter avere un pieno controllo dell'intero ciclo di vita del progetto. Per raggiungere questi obiettivi bisogna comprendere completamente le richieste del cliente, pianificare le risorse necessarie, organizzare in maniera precisa e puntuale i tempi di esecuzione e verificare ogni singola fase. In questo articolo inizieremo a vedere insieme alcuni casi di progettazione ed esperimenti di rapid prototyping grazie ai quali comprenderemo meglio quali sono le sfide principali cui si va incontro durante la realizzazione di un progetto. Il tutto, ovviamente, mediante esempi pratici e sperimentazione in casi reali. Ciò indicherà la strada, a partire dal più semplice fino al più complesso progetto IoT. Siete pronti?*

## INTRODUZIONE

La progettazione è probabilmente una delle mansioni più complesse e variegate che esistano. A prescindere da ciò che si sta progettando, infatti, ad un progettista è sempre richiesto di avere una visione d'insieme completa, puntuale e capace di tener conto e traccia di ogni singola problematica, del perché è emersa e soprattutto di come è stata eventualmente risolta. L'evoluzione tecnologica, ma anche quella della conoscenza, ha consentito alla tecnica di diventare sempre più rapida ed efficace, tuttavia le esigenze di base non sono cambiate. In questo contesto si afferma il cosiddetto **rapid prototyping**, un insieme di tecniche, strumenti e procedure che puntano ad accelerare ed ottimizzare parti del processo di progettazione. Prima di arrivare a questo, però, si sono attraversate diverse fasi storiche che è importantissimo comprendere e conoscere.

## LE RIVOLUZIONI INDUSTRIALI

L'automazione ha ragioni storiche e la si può convenzionalmente far risalire alla fine del 1700 con l'introduzione della macchina a vapore all'interno dell'industria tessile nell'antica Gran Bretagna. All'epoca i lavori di tessitura, che erano tra i lavori di manifattura più pregiati e ricercati nel mondo, venivano prevalentemente fatti a mano o con l'ausilio di macchine che però avevano meccanismi di automazione davvero rudimentali.

La macchina a vapore cambiò tutto. L'invenzione di un meccanismo automatico che sfruttava la forza del vapore apparve in quel momento storico davvero rivoluzionario.

Certamente gli uomini del tempo pensarono a quanto queste macchine potessero essere veloci e quanto potessero fare di più rispetto a prima. Molto probabilmente non gli fu chiaro fin da subito che stavano per cambiare il mondo in una maniera tale che indietro non si sarebbe mai più tornati.

Agli inizi del '900, in una fabbrica di automobili americana, Henry Ford decise di mettere in pratica il concetto di catena di montaggio.

Gli operai hanno dei compiti ben precisi, li eseguono in sequenza, manipolano solo quello che si trova davanti a loro e la postazione, piuttosto che la fabbrica in quanto tale, diventa il loro luogo di lavoro. Vengono diminuiti gli spostamenti, ottimizzata la logistica, assegnati staticamente i compiti. È l'inizio della produzione di massa.

Da quel momento in poi, gli storici individuano un incremento sempre maggiore della velocità di espansione e di ottimizzazione degli strumenti automatici all'interno della nostra società. L'industria non si basa più sull'agrario ma sulla manifattura, l'economia diventa prevalentemente di settore terziario e di servizi. Fino ai giorni nostri, quando l'industria manifatturiera incontra il mondo digitale. Nasce cosi' il concetto di **digitalizzazione dei processi produttivi** e la tecnologia cambia ancora.

***"E qui arriviamo ai giorni nostri, in cui si parla di Industry 4.0. Ma cos'è?***

Con il termine "Industry 4.0" si fa riferimento ad un programma quadro lanciato dal governo tedesco nell'ambito di un progetto di digitalizzazione dell'intero comparto industriale europeo da parte dell'Unione Europea. Lo scopo è quello di seguire alcune direttrici nel processo di sviluppo:

1. innovazione di processo, ovvero ammodernamento, digitalizzazione, ottimizzazione continua di tutto quello che ha a che fare con ciascun processo industriale. Gestione del magazzino, approvvigionamento merci, spostamenti e logistica, assemblaggi, test, verifiche, anche di qualità;
2. innovazione di prodotto, ovvero adattamento tecnologico dei prodotti industriali al fine di incontrare i più aggiornati criteri in termini di compatibilità ambientale, eliminazione di sostanze pericolose e standard di funzionamento e sicurezza;
3. complessiva riduzione delle emissioni di CO2 e di altre sostanze pericolose nell'atmosfera entro il 2020;
4. utilizzo di fonti energetiche rinnovabili per raggiungere l'abbattimento dei costi di produzione e generazione di energia.

Questi, e molti altri ancora, rappresentano gli assi portanti della rivoluzione digitale che abbiamo di fronte a noi e che sono oggi il vero motore propulsore dell'innovazione e del rinnovamento che stiamo osservando.

## IL PROCESSO INDUSTRIALE

Il processo industriale è, in via del tutto generale, descrivibile identificando alcune fasi fondamentali:

1. il pensiero, ovvero il momento in cui viene immaginato e descritto ed, in via preliminare, progettato il sistema o la soluzione di riferimento. Probabilmente si tratta della fase più importante, quella in cui tutto ciò che il sistema si pensa debba poter fare, viene delineato. In generale, in questa fase il sogno comincia a prendere forma. Tutto parte, di solito, da un'idea, ma il pensiero è probabilmente la fase operativa nella quale qualcosa di cui fino a quel momento si era soltanto parlato, comincia a prendere forma per diventare reale;
2. il progetto, ovvero il momento in cui la semplice descrizione non basta più e bisogna cominciare operativamente a stabilire numeri e intervalli di interesse, in definitiva fare i conti. In questa fase tutte le specifiche tecniche e funzionali cominciano a diventare concrete e si realizza quel compromesso tra l'idea fantasiosa iniziale ed il prodotto vero e proprio che verrà creato. In

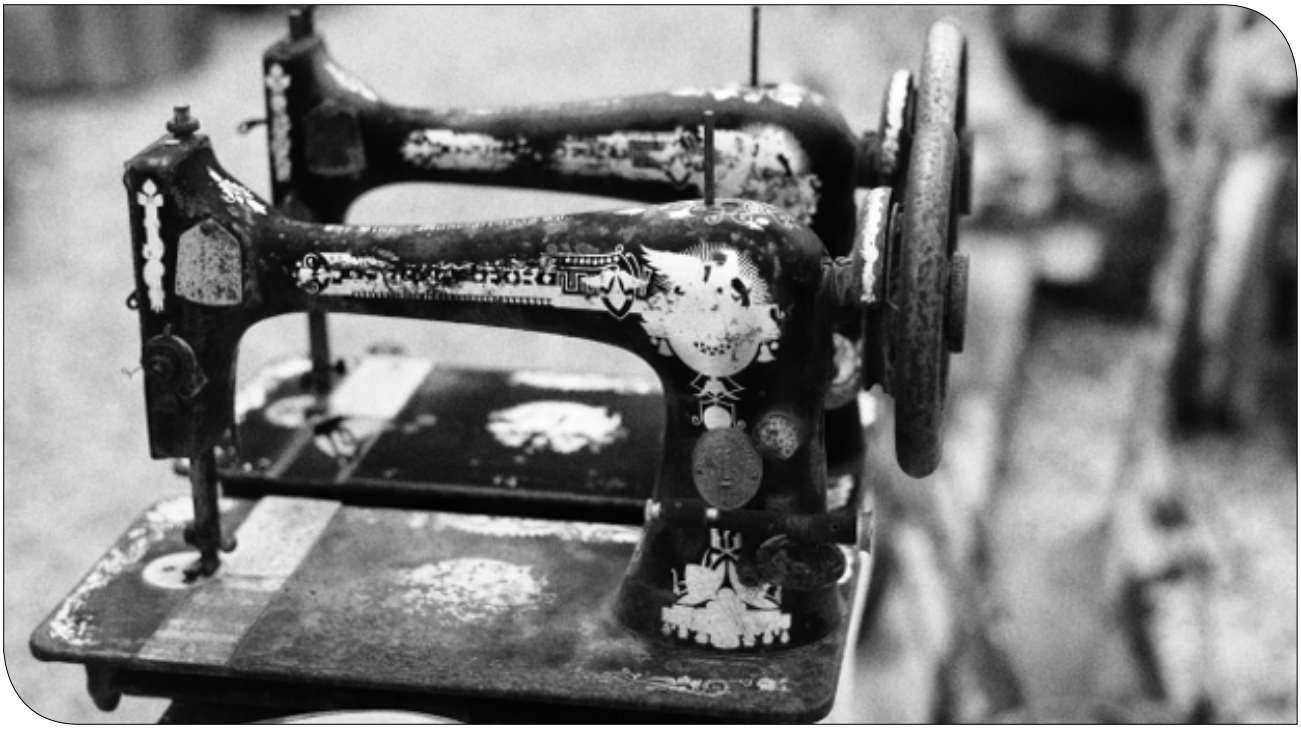


*Figura 1. Macchine da cucire d'epoca*

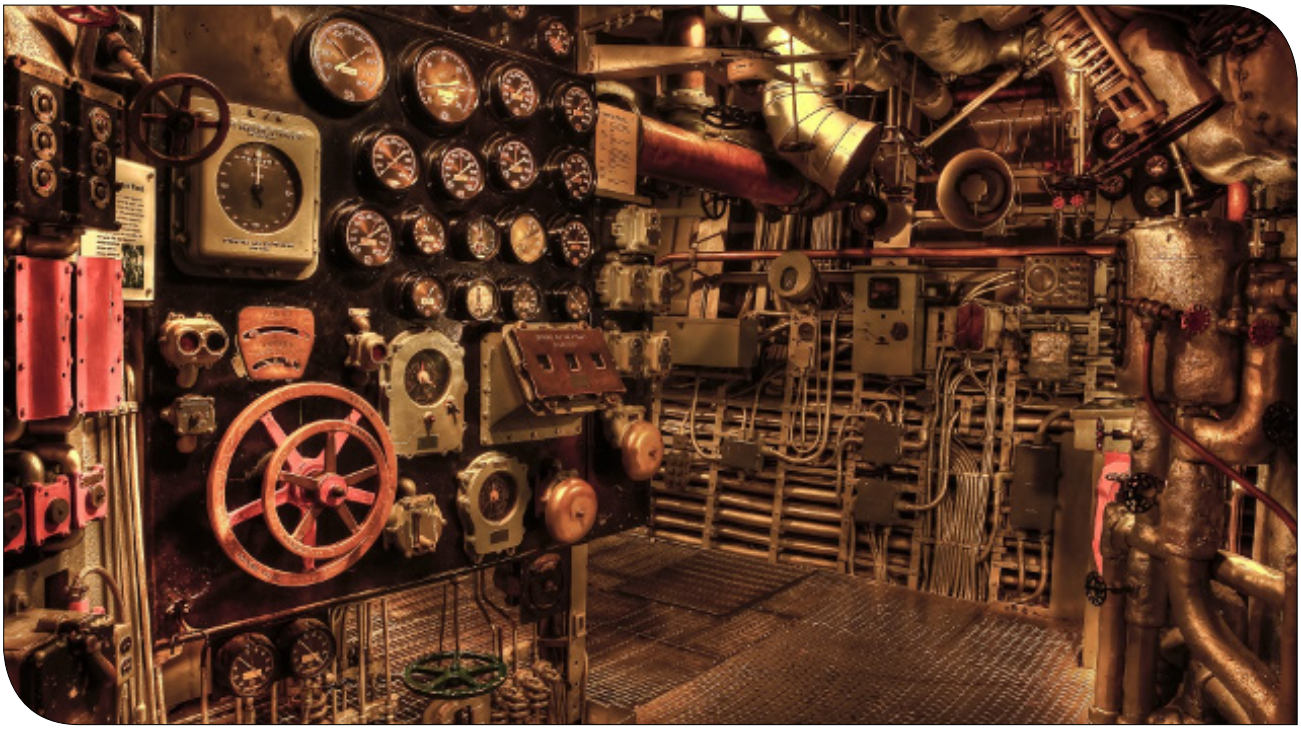


*Figura 2. Macchinari e valvole di controllo all'interno di una fabbrica*

questa fase è necessario essere estremamente concreti, disponibili al compromesso ma soprattutto adattabili a concertare le varie esigenze;

3. la fabbricazione, ovvero la fase in cui uno stabilimento fatto da persone, oggetti, materie prime e strumenti di lavoro, diventa la fucina della nostra idea. Qui tantissime persone, spesso non consapevoli di quello che stanno facendo nel complesso, eseguono compiti che contribuiranno a tirar fuori il prodotto finito, ovvero contribuisce a creare un oggetto tramite azioni elementari come avvitare una vite o controllare, mediante ispezione visiva, che un pezzo sia lucido o del giusto colore;
4. la distribuzione, che naturalmente è una fase che cambia tantissimo in funzione di cosa stiamo producendo. Oggi ad esempio, non c'è più bisogno di recarsi fisicamente nei negozi per comprare un software su un supporto fisico come un CD-ROM. L'evoluzione tecnologica ha garantito connessioni Internet molto veloci anche per utenze domestiche e quindi è possibile scaricare una copia del software in digitale mediante download diretto grazie ad un sito dedicato. In realtà, anche per prodotti fisici si potrebbe fare lo stesso ragionamento, dal momento che esistono tante aziende che si occupano di effettuare spedizioni in ogni parte del mondo in tempi brevi, entro 24 o al massimo 48 ore.

La domanda, dunque, può diventare: cosa c'è da ottimizzare in tutto questo? Le risposte, in verità, sono tantissime, perché è possibile ottimizzare tempi, costi e prototipi intesi come risultato delle singole fasi intermedie di prototipazione che il progetto attraversa. Naturalmente la semplice voce "tempi", vuol dire poco. Per tempi si intendono, ad esempio, i tempi di assemblaggio, di approvvigionamento delle merci e di spedizione. Ma si tratta anche dei tempi inerenti alle procedure all'interno di una fabbrica. Anche la voce costi è polivalente, perché il costo non è semplicemente l'ammontare di denaro speso per l'acquisto delle materie prime ma può essere anche il numero di persone necessarie affinché un singolo processo venga completato. Tra i costi non incide solo il capitale umano, ma anche la pulizia di uno stabile, le ispezioni e le verifiche per il controllo di qualità e, non ultimo, l'adeguamento a nuove normative. Per ora, quindi, stiamo parlando in linea generale. In base allo specifico prodotto, processo, azienda o prototipo, sarà possibile declinare in maniera precisa e puntuale tutte queste voci, nel dettaglio.

## VIRTUAL PROTOTYPE

Quanto detto fino a questo momento riguarda, in generale, la prototipazione, la progettazione e l'impianto industriale così come lo conosciamo oggi. Naturalmente,

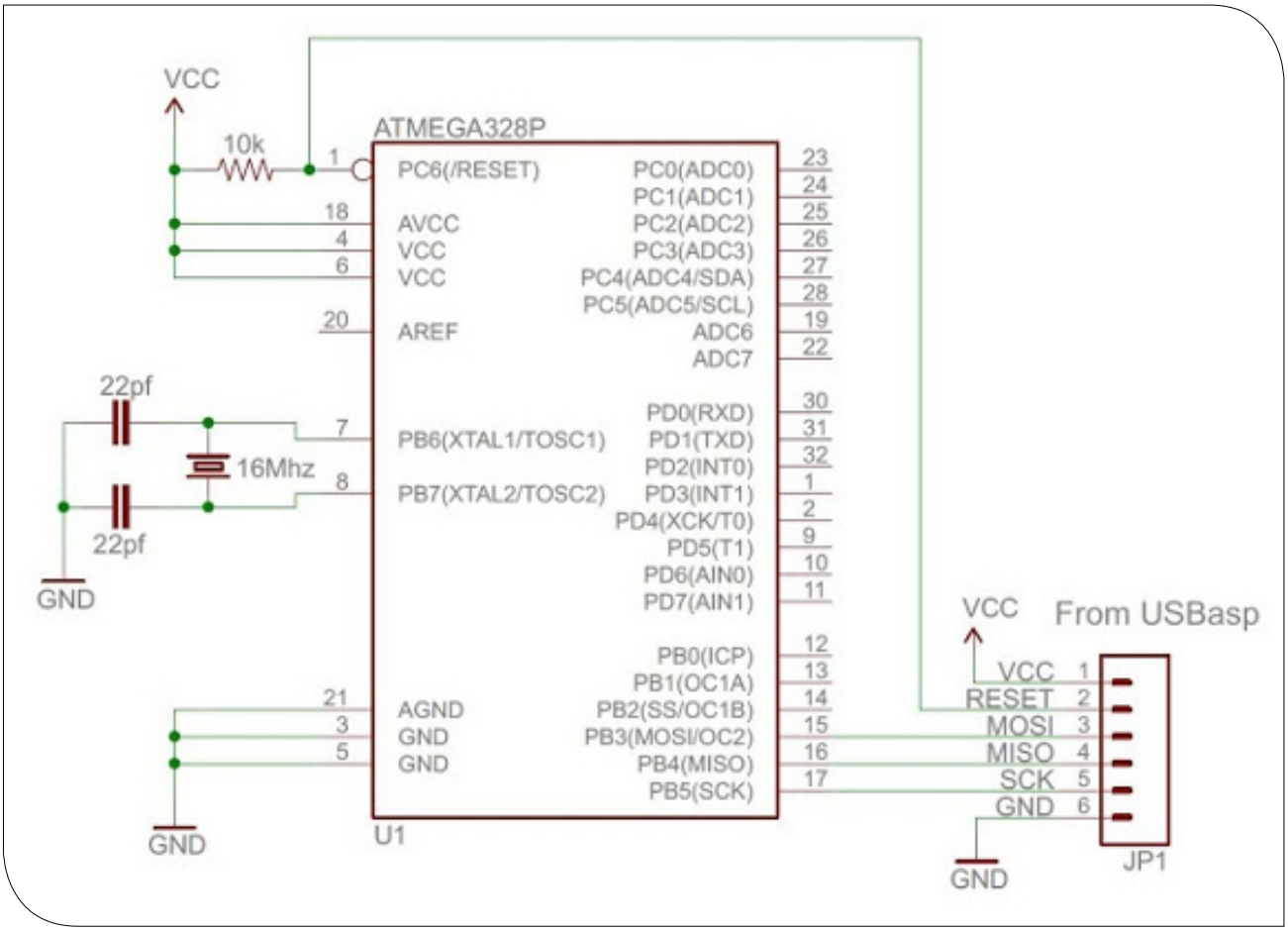



*Figura 3. Configurazione minima per l'avvio di un ATMega328*

però, il nostro interesse quando parliamo di tecnologie di rapid prototyping è molto più specifico. Ad esempio, la nostra comunità su Elettronica Open Source è prevalentemente composta da ingegneri, hobbisti, makers che si occupano di prototipazione elettronica, informatica e programmazione più in generale. Questo tipo di figure professionali spesso ha a che fare con il concetto di **prototipo virtuale**, altrimenti noto come **digital mock-up**. È indubbio che creare un prototipo significa non andare sul mercato con la prima realizzazione di un progetto. Esso sicuramente avrà problemi di funzionamento, anche di ordine pratico, sarà poco utilizzatabile, probabilmente non avrà un design particolarmente accattivante. Il concetto della prototipazione e della realizzazione di un prototipo virtuale, dipende dal fatto che è necessario che un prodotto possa essere presentato, analizzato e testato relativamente ai parametri fondamentali inerenti il suo processo. In particolare, ci stiamo riferendo a:

- progettazione
- ingegnerizzazione
- manutenzione
- smaltimento

È importante comprendere che su un prototipo virtuale è possibile effettuare delle operazioni che servono a testare l'oggetto virtuale come se fosse un oggetto fisico. La scelta di strumenti software opportuni fa parte di quelle voci di design di cui abbiamo parlato in precedenza. Nel concreto, **se vogliamo creare una scheda elettronica integrata che si occupi di elaborare dei segnali, il nostro prototipo virtuale molto probabilmente sarà realizzato in ambiente CAD elettronico**, grazie a strumenti come KiCAD, OrCAD, ma anche EAGLE, Altium Design e simili. Se vogliamo sviluppare una soluzione software per la gestione del database di un sistema di accessi controllato, il nostro prototipo virtuale sarà in realtà un codice lungo un certo numero di righe, scritte in un determinato linguaggio e che verrà redatto da uno sviluppatore in un IDE come Visual Studio, Eclipse o altri.

## OPEN SOURCE MANUFACTURING

L'Open Source è in generale una filosofia, ma rappre-

senta nel concreto una vera e propria opportunità per le aziende. L'esempio principale di questo è il sistema operativo Mac OS di Apple. Interamente basato su Linux, il sistema operativo si avvantaggia dell'esperienza decennale degli utenti della community Open Source per creare una soluzione proprietaria, venduta a costi sostenuti, che però viene ottimizzata in house. Apple ha ereditato l'intero know-how della comunità globale dell'Open Source che si è occupata di Linux dall'inizio fino ad oggi per poter partire avvantaggiata e creare una serie di applicazioni e servizi che venissero specificatamente ottimizzati. Il vantaggio di avere Mac OS non è soltanto questo, dal momento che l'ecosistema Apple gode di un'ottima fase di sviluppo condotta da tutti gli ingegneri informatici che all'interno dell'azienda hanno continuativamente mantenuto la soluzione, implementando funzionalità via via più avanzate. Esistono altri esempi di come l'Open Source rappresenti una vera e propria opportunità commerciale. In generale, quello che possiamo dire è che la produzione industriale che si basa sui prodotti della comunità Open Source di solito si avvantaggia anche di una community.

Quando un progetto viene reso disponibile on-line, la creazione della community di utenti è davvero un valore aggiunto. Gli utenti che scrivono sui forum, rispondono sui blog oppure creano dei progetti derivati, rappresentano una vera e propria risorsa non soltanto perché le aziende possono fare scouting di persone talentuose, ma anche e soprattutto perché questo tipo di interazioni funziona quasi meglio del beta testing dal momento che gli utenti rispondono ed interagiscono su specifiche problematiche.

Oltre alla risoluzione di problemi ed alla loro individuazione in via preventiva, vengono anche suggerite le possibili espansioni del software. Quali funzioni mancano? Quali gli utenti preferirebbero?

Quali sviluppare prima e dopo? Sono tutte domande importantissime alle quali un'azienda deve rispondere. Molto spesso, per farlo, è indispensabile fare indagini di mercato, coinvolgere personale delle risorse umane, interagire con un campione di utenti, parlare con chi si occupa del marketing e così via dicendo.

Una community di utenti snellisce questo processo, abbatte i costi di realizzazione e propone anche altri vantaggi minori.

## DALLA TEORIA ALLA PRATICA

***Making Hardware***

Nella **prototipazione rapida di sistemi elettronici integrati**, come ad esempio schede embedded, le sfide progettuali si possono affrontare in fasi successive. Prima di tutto, infatti, è indispensabile comprendere quali sono le specifiche di progetto. Dall'idea è necessario derivare tutte le caratteristiche funzionali, i tipi di interazioni con il sistema, le tipologie di segnali ma anche le

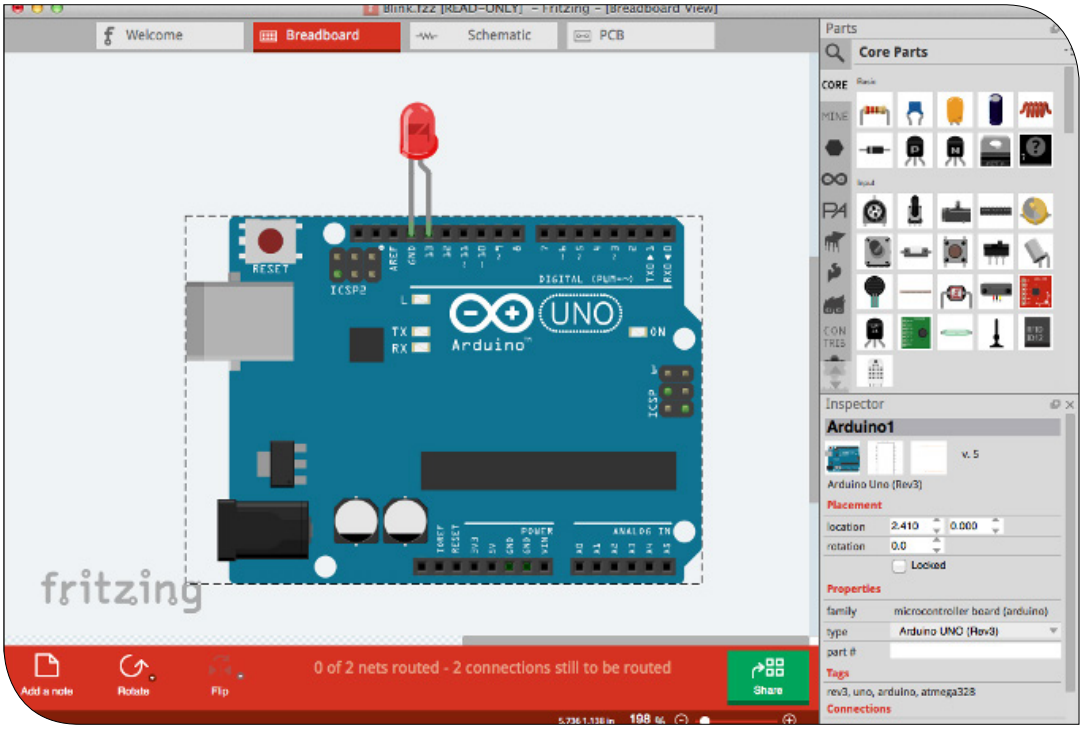



*Figura 4. Virtual Mock-up con Fritzing*

possibilità di espansione. La prima fase, ovvero quella della descrizione generale del progetto, in cui ciò che immaginiamo che il progetto dovrà fare viene scritto su carta, viene seguita da una fase di stesura delle specifiche tecniche del sistema. Una volta definito l'insieme di tutte le specifiche tecniche e funzionali che il sistema deve poter garantire, sia lato utente nell'interazione, sia lato sviluppatore per la possibilità di espandere la scheda, ovvero montare nuovi componenti piuttosto che collegare periferiche esterne, si passa alla scelta dei componenti. Questa fase è davvero critica perché la scelta di un componente condiziona moltissimo non soltanto il fatto che si rispettino le specifiche correnti, ovvero, della versione del sistema che si vuole realizzare, ma anche e soprattutto la possibilità di espandere le funzionalità nel futuro. Prendiamo come esempio il caso di Arduino. Nella sua versione Uno, la scheda prevede l'utilizzo di un microcontrollore, in particolare l'ATMega328p, il quale viene collegato ad uno zoccolo (socket) che a sua volta accetta collegamenti verso: una tensione di alimentazione, ovvero un collegamento duale al potenziale più alto ed al potenziale di massa, che naturalmente nei circuiti digitali in logica TTL 5 V è sempre 0, una serie di collegamenti ancora rimasti fluttuanti per le periferiche di input/output (I/O) esterne, alcuni altri collegamenti per l'interfacciamento tramite seriale e quindi il collegamento USB. La configurazione minima che consente la messa in funzione di Arduino nella sua versione Uno è la seguente:

Ovviamente, in questa versione, la scheda non è assolutamente bella da vedere, certamente non è ingegnerizzata ma propone le specifiche iniziali di progetto della scheda ovvero mettere in funzione un microcontrollore. Questo è un chiaro esempio di mock-up del progetto dove il progetto è, a sua volta, una **scheda di prototipazione che diventi la base per mock-up di progetti elettronici**. Sembra un paradosso, ma vale la pena sottolineare che il ruolo centrale della scheda a microcontrollore obbliga ad eseguire tutti i passi di progetto di cui abbiamo parlato fino a questo momento per essere sicuri che la scheda, nella sua versione finale, funzioni esattamente come deve. Completato il mock-up, si passa al test ed alla verifica, ovvero si provano le funzionalità di base, si verificano i livelli di tensione e quelli di corrente. Dopo aver fatto questo, è indispensabile procedere nell'aggiunta successiva di funzionalità e, pertanto, vale la pena di provare i singoli collegamenti sui singoli pin, in funzione della loro specifica. **Una volta realizzato il circuito in vitro, ovvero su breadboard, in fase prototipale, si può passare alla formalizzazione dello schematico su CAD**. Ovviamente la scelta del CAD di riferimento è molto importante e soprattutto è soggettiva. I progettisti che si occupano di sbroglio hanno molte preferenze in tal senso e scelgono in maniera elettiva il loro CAD di riferimento per potersi specializzare nel suo utilizzo, personalizzare le routine per effettuare lo sbroglio ed impostare l'ambiente di lavoro così da ottimizzare la loro produttività ed in ultimo il processo. Dalla realizzazione del circuito, poi, si passa allo sbroglio vero e proprio in cui tutti e soli i componenti che servono per implementare le funzionalità previste vengono messi insieme, disposti in maniera tale da occupare il minor spazio possibile, connessi ottimizzando al meglio la lunghezza ed il numero dei collegamenti. A questo segue la fase di realizzazione vera e propria della scheda. Se non volete utilizzare delle tecniche fai da te, nonostante la bellezza del Do-It-Yourself, potete certamente rivolgervi a servizi online per un preventivo. **Una volta che avrete realizzato un circuito stampato, sarà la volta del posizionamento dei componenti**. Questa è una fase molto critica e deve essere eseguita con grande precisione, soprattutto quando le dimensioni dei circuiti e degli integrati di riferimento diminuiscono. **L'elettronica integrata ha dato una notevole spinta ed un grande impulso allo sviluppo di tecniche avanzate per la gestione di componenti microelettronici**. Questo vuole anche dire che risulta difficile creare le proprie schede a casa, ovvero andare oltre la fase dello sbroglio. Quando la scheda è completa, montata, ripulita da eventuali residui della saldatura, può essere testata. Il testing deve essere fatto verificando ogni singola funzionalità, ogni possibile interazione e, possibilmente, andando anche oltre quelli che sono i limiti funzionali imposti.
Ad esempio è indispensabile testare i livelli di tensione di funzionamento effettivamente supportati perché condizioni di sovralimentazione devono poter essere documentate. Le domande alle quali rispondere arrivati a questo punto sono:

- in che condizioni la scheda smette di funzionare in maniera ottimale?
- quali sono i possibili errori dovuti alle condizioni di malfunzionamento?
- quali sono tutte le condizioni di malfunzionamento possibili?
- ci sono particolari precauzioni da utilizzare?

Tutte queste domande servono per poter scrivere una sorta di manuale.

| Maker | Ingegnere |
|---|---|
| Impara sul campo tutto quello che gli serve | Ha solide basi teoriche |
| Fortunatamente c'è Internet | … già! |
| Sperimenta anche praticamente | Molto spesso ha accesso solo a simulazioni |
| È più giovane | Beh, una laurea in ingegneria porta via del tempo… |
| Ha bisogno di provare più volte | Può simulare N volte e produrre andando a colpo «sicuro» |
| Non si preoccupa di protocolli e standard | Sa bene quanto contano ed è in grado di valutarli preventivamente |
| Non ha quasi idea di cosa sia la certificazione | Sa quantificare costi e processi per la standardizzazione |

***Making software***

Con il software la cosa procede in maniera non dissimile ma forse appena più facile da visualizzare e comprendere.
Se il focus è l'interazione con l'utente, il mock-up di un progetto software, infatti, molto facilmente potrebbe essere il front end di un portale web più in generale. Il mock-up di software CAD potrebbe essere l'interfaccia con la quale l'utente interagisce per azionare i vari comandi oppure accedere alle varie funzioni, come una dashboard.
Naturalmente, in un mock-up con questa funzione, si sottintende che tutti i pulsanti, tutte le interazioni e tutti i servizi sono gestiti da un corrispondente servizio operante nella sezione di back end.
Un progetto software attraversa le stesse fasi della prototipazione hardware e differisce da essa perché la fase di test e validazione è spesso più rapida.
Quando si realizza un sito web, si vuole creare un mock-up per la realizzazione di una landing page, molto facilmente si può aggiungere del codice HTML oppure modificare il codice esistente direttamente da un template preimpostato, verificare il risultato atteso e poi codificare, in egual misura, all'interno dei file di configurazione del sito stesso.
Per questi motivi nella ***prototipazione software*** si attraversano fasi estremamente simili a quelle necessarie nella prestazione hardware e che si possono riassumere come segue:

- analisi e caratterizzazione dello scenario operativo
- specifiche funzionali di progetto
- analisi del budget e caratterizzazione distinta delle risorse economiche, temporali ed umane necessarie
- analisi dei linguaggi di programmazione e degli strumenti più idonei per la realizzazione
- nuova stima dei costi e verifica degli obiettivi di medio termine
- sviluppo nella prima versione rilasciabile
- test

Molte di queste richiedono l'interazione diretta con il cliente, il suo feedback costante sulla disponibilità economica e la continua verifica delle risorse a disposizione.
Quando si deve sviluppare un progetto software è davvero importantissimo comprendere quale sia il linguaggio di programmazione più idoneo per poter svilupparlo, e non è neanche detto che ne basti uno.
La ricchezza delle librerie, la documentazione, il supporto da parte di diversi IDE e la portabilità, sono soltanto alcuni dei criteri progettuali che consentono di discer-

nere se uno strumento software sia più o meno idoneo. È importante comprendere, ad esempio, se è necessario utilizzare un database per popolare un sito internet e quando questa parte dello sviluppo debba avvenire.
L'utilizzo di strumenti che consentono lo sviluppo in PHP, piuttosto che in CSS, HTML5 o altri linguaggi specificatamente pensati e sviluppati per il Web, può non essere il core business di ciò che deve essere realizzato.
L'interfaccia web potrebbe ad esempio essere un plus del progetto che magari, nella fase prototipale, durante la realizzazione del mock-up, non è indispensabile.
È possibile se si sta lavorando nell'ambito della telemedicina, che le funzionalità software principali da sviluppare siano inerenti all'interazione del medico con il sistema e quindi piuttosto che un'interfaccia web è importante sviluppare un'interfaccia software locale che consenta al medico di elaborare i dati dopo una visita. Il progetto software potrebbe basarsi sull'interazione Uomo-Macchina del paziente durante la visita; in quel caso, lo sviluppo software sicuramente non avrà a che fare con il popolamento di una tabella all'interno di un database ma piuttosto sarà importante utilizzare un prototipo scritto in Python per gestire il Raspberry Pi di turno che ha il compito di acquisire i dati dal sistema che comunica tramite Bluetooth.
E' possibile che ci siano da sviluppare dei driver specifici per la particolare interfaccia di comunicazione e quindi l'effort maggiore dovrà essere speso per scrivere una libreria, magari in C.
Che il vostro prototipo sia hardware o software, tutte queste esigenze possono essere codificate e messe in relazione tra loro secondo un sistema di priorità. Ne risulta, quindi, il fatto che la prototipazione rapida deve necessariamente prevedere delle fasi di:

- Modellazione
- Conversione
- Controllo e preparazione
- Costruzione
- Post-processing

## SOLUZIONI SOFTWARE DI SUPPORTO

Tutti i processi descritti non devono per forza essere svolti in rigide fasi differenti con soli strumenti differenti. Il mondo dell'Open Source oggi offre una serie di possibilità davvero interessanti che possono consentire di convertire il processo e semplificarne alcuni passaggi. In particolare, vi abbiamo parlato di prototipazione hardware e software come se dovessero essere fasi differenti, ma alcuni strumenti di prototipazione rapida riescono a dare la dimensione che non ci sia più una rigida distinzione tra queste fasi. **Una scheda come Arduino consente di accelerare vertiginosamente le fasi di test, modificando il numero di sensori oppure il numero di attuatori, verificando al volo se un LED sia più idoneo, o comunque sufficiente, rispetto all'utilizzo di una serie di LED**. Un esempio di software che mette insieme le esigenze di progettisti che stanno sperimentando con schede di prototipazione come Arduino, è Fritzing.
Si tratta di un framework di lavoro che consente contemporaneamente di effettuare delle ricerche e consultare un blog di informazione. L'interazione con il portale on-line di Fritzing consente l'utilizzo di una banca dati davvero vasta e variegata.
Tantissimi makers, hobbisti, progettisti più o meno esperti, interagiscono con la community per creare nuove proposte di nuovi progetti, sempre più complessi. Ma non finisce qui, perché tra le cose che ci consente di fare vi è la **prototipazione hardware**, mettendo a disposizione tutta una serie di librerie per le singole schede, piuttosto che per i componenti.
Fritzing, infatti, consente al progettista di creare un prototipo dei collegamenti. Il mock-up virtuale, che è in realtà il vero prodotto che si realizza tramite Fritzing, viene utilizzato non soltanto per valutare il numero di collegamenti, ma anche dal punto di vista della disposizione degli stessi.
Si possono modificare i componenti, provare a sostituire le varie schede, ma anche rispondere ad alcune domande, come ad esempio: quella particolare scheda della famiglia Arduino è sufficiente?
Risponde alle mie esigenze?
Troppo spesso, quando non si è molto esperti, si può sottovalutare il numero di pin necessari per effettuare dei collegamenti, per esempio dovendo collegare un display LCD.
Grazie a Fritzing è possibile verificare fin da subito se il numero di pin disponibili è sufficiente. Si può provare ad alloggiare il prototipo virtuale su più breadboard oppure modificare il colore dei fili per poter distinguere visivamente, in maniera semplificata, se il collegamento sia relativo ad una linea di alimentazione oppure al trasferimento dei dati. Il fatto che ci sia un ampio supporto per diversi componenti hardware dipende dal fatto che è un progetto supportato da moltissime aziende.
Anche della famiglia Raspberry Pi, ad esempio, all'interno di Fritzing è possibile selezionare diverse versioni della scheda.
Per tutti i makers che abbiano fatto acquisti online per i loro progetti, probabilmente SeedStudio, o magari Spar-

kfun, non sono solo nomi già sentiti, ma veri e propri fornitori di fiducia. Bene, i loro prodotti sono praticamente quasi tutti inseriti all'interno del catalogo e comunque sono importabili.
Quando avete finito di effettuare i collegamenti, simulando la presenza di una breadboard, potete passare a fare quello che davvero fa un progettista, ovvero, creare lo schema elettrico. Tutti i componenti che prima erano rappresentati in maniera fumettosa, adesso diventano dei blocchi, ovvero circuiti integrati, in package isolati, con un certo numero di piedini di interfaccia che sono funzionalmente diversi e quindi devono essere collegati in maniera precisa.
Fritzing è un ambiente in cui anche il progettista meno esperto riuscirà a tirar fuori uno schema elettrico. In questa fase la sfida per tutti sarà quella di minimizzare il numero dei collegamenti ma soprattutto ottimizzare la loro disposizione. Il risultato ottenuto è l'input per la fase di sbroglio, ovvero quella in cui il progettista realizza i circuiti e prepara i file da esportare per poter realizzare un circuito stampato sul quale poi i componenti verranno saldati. Anche qui Fritzing funziona egregiamente, dando, seppur per piccoli progetti, una grossa mano al progettista meno esperto. Non aspettatevi di tirar fuori dei progetti professionali, stiamo sempre parlando di semplici prototipi. Ma è un inizio. Tutte queste informazioni probabilmente le avevate già. Ma se state valutando un percorso imprenditoriale di formazione e di riqualificazione del vostro capitale umano, ecco che queste considerazioni assumono decisamente tutta un'altra veste!
In conclusione di questo articolo, è interessante notare come la prototipazione rapida accomuni gli ingegneri, i professionisti, coloro che hanno maturato tanti anni di esperienza sul campo, ma anche i makers, gli hobbisti e quelli che cominciano. Ci sono molti punti di tangenza ed anche alcune differenze fondamentali. In questa tabella li riassumiamo in maniera sintetica.
I punti di tangenza riguardano il metodo.
Quali sono le problematiche e come possono essere intuite le soluzioni, rappresenta l'attitudine al problem solving che è presente in entrambe le categorie.
Ciò che distingue un professionista da un dilettante è il fatto che il metodo e l'esperienza si affinano nel corso del tempo.
Entrambe queste categorie concordano sul fatto che Internet sia una grandissima risorsa, ciò che li distingue è il modo in cui lo utilizzano. Per un professionista, Internet è un tavolo di confronto, una fonte di informazioni che contiene ben più di quanto effettivamente gli serva. Il dubbio tipico del professionista è specifico e caratterizzato, la sua capacità di ricerca della risposta alla domanda è fine e riesce a scartare con metodo tutto ciò che non risponde esattamente alla sua domanda. Un dilettante utilizza invece Internet come un terreno fertile dal quale provare ad attingere tutto quello che può, la disponibilità delle informazioni è la chiave che rende la sua inesperienza velocemente meno influente. Man mano che legge, un dilettante comincia a capire quali sono le fonti più attendibili e complete e guadagna confidenza anche con i temi trattati.
Ci vogliono anni per trasformarsi in un professionista ma, naturalmente, la disponibilità delle informazioni, anche in caso di ridondanza, rappresenta una ricchezza. Il dilettante può affinare il suo spirito critico, riconoscendo le fonti che sono più o meno dettagliate, attendibili e ricche.
Un esempio che accomuna tutti coloro che fanno sviluppo software è Stack Overflow, un portale che funziona davvero molto bene per chiunque sviluppi.
Ci sono domande di ingegneria del software su funzioni specifiche di librerie specifiche che possono essere utilizzate in alcuni contesti ma ci sono anche domande di carattere generale.
La community è un elemento chiave su questo portale perché vota le risposte più pertinenti, più complete e più utili.
L'esperienza su Stack Overflow è comprovata dal numero e dal tipo delle risposte. Il sistema è abbastanza meritocratico e rappresenta sicuramente un ottimo punto di riferimento per moltissimi neofiti di linguaggi di programmazione per il Web ma anche per tutti coloro che iniziano con Python.

## CONCLUSIONI

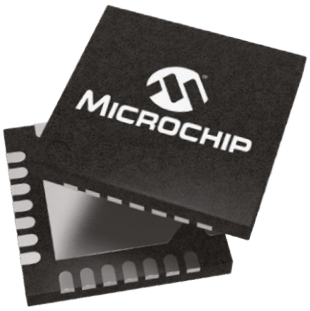
Questo nostro primo appuntamento con il making e l'IoT termina qui. Per ora abbiamo ragionato insieme dei concetti preliminari e di cosa serva per iniziare a progettare e fare concretamente l'IoT.
Nel prossimo articolo studieremo tutte le componenti in dettaglio, analizzeremo le interfacce di comunicazione e completeremo il quadro informativo per poi caratterizzare un paio di progetti davvero smart. Alla prossima.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/iot-e-chi-iot-lo-fa-dallidea-al-prototipo

# Collegati

## Collegati al mondo, con o senza fili

Durante la tua giornata devi affrontare molte sfide. In Microchip lo sappiamo, e ti rendiamo facile l'aggiungere connettività al tuo progetto. Sia che tu abbia bisogno di una solida e affidabile connessione cablata o della mobilità e convenienza del wireless, l'ampio portfolio Microchip ti aiuterà a stabilire facilmente la connessione.




Per una ulteriore semplicità, i nostri MCU e MPU sono progettati per essere compatibili con i nostri dispositivi wired e wireless. E possiamo aiutarti ad arrivare rapidamente sul mercato con moduli certificati e stack di protocollo production-ready.

Connettiti con Microchip e scopri come connetterti in modo sicuro al mondo che ti circonda.

SMART | CONNECTED | SECURE

**Effettua la connessione su**
**www.microchip.com/Connected**

MICROCHIP

Il nome e logo Microchip ed il logo Microchip sono marchi industriali registrati di Microchip Technology Incorporated negli U.S.A.e altri Stati. Tutti gli altri marchi industriali appartengono ai rispettivi titolari registrati.
© 2019 Microchip Technology Inc. Tutti i diritti riservati. DS00002768A. MEC2231A-ITA-11-19

# IOT È CHI IOT LO FA: COMPRENDIAMO GLI STRUMENTI

di **Pietro Boccadoro**

*Nello scorso appuntamento ci siamo occupati di comprendere cosa voglia dire iniziare la prototipazione in ambito IoT. Inquadrare le specifiche tecniche e progettare in maniera chiara fin da subito, significa spendere del tempo per fare attività di brainstorming, ma il risultato che si ottiene è la comprensione totale di quanto si desidera raggiungere. Chiariti gli obiettivi, è fondamentale focalizzare la propria attenzione sugli strumenti. Quali sono quelli più idonei? Come si possono scegliere? Dove posso trovarli? Come li metto in relazione tra loro? In questo articolo vedremo uno studio specifico sulle schede di prototipazione rapida, la scelta dei sensori ed il loro interfacciamento. Siete pronti?*

## INTRODUZIONE

Fare prototipazione rapida utilizzando strumenti e tecnologie, oggi ben diffuse e mature, significa innanzitutto lavorare con dati digitali che vanno opportunamente processati. Ciò implica analizzarli, elaborarli e poter proporre in uscita dal nostro sistema un qualche risultato che sia verificabile, misurabile, ripetibile ed accurato. È facile immaginare che queste funzioni siano integrate all'interno di un dispositivo e realizzate tramite circuiti. Ma come deve funzionare esattamente? Come può funzionare? Quali sono le differenze tra le varie implementazioni possibili? Che tipo di logica deve poter gestire il nostro componente? Rispondere a domande di questo tipo è molto più importante di quanto non si pensi perché le risposte influenzano notevolmente le prestazioni attese e i risultati finali.

## MICROPROCESSORE VS MICROCONTROLLORE

La prima esigenza è sicuramente quella di gestire i dati. È importante riprendere i concetti di microcontrollore e microprocessore ed estenderli in maniera tale da fornire un quadro informativo più chiaro che consenta di distinguere tra processori, controllori e DSP. Soprattutto per chi è alle prime armi, probabilmente ci sarà un po' di confusione all'inizio sulla distinzione tra queste entità. **Il microprocessore è un dispositivo integrato che funziona grazie ad una programmazione ben precisa**, si impostano una serie di istruzioni che vanno eseguite in sequenza per effettuare una specifica operazione o una serie di operazioni sui dati. In pratica, la programmazio-

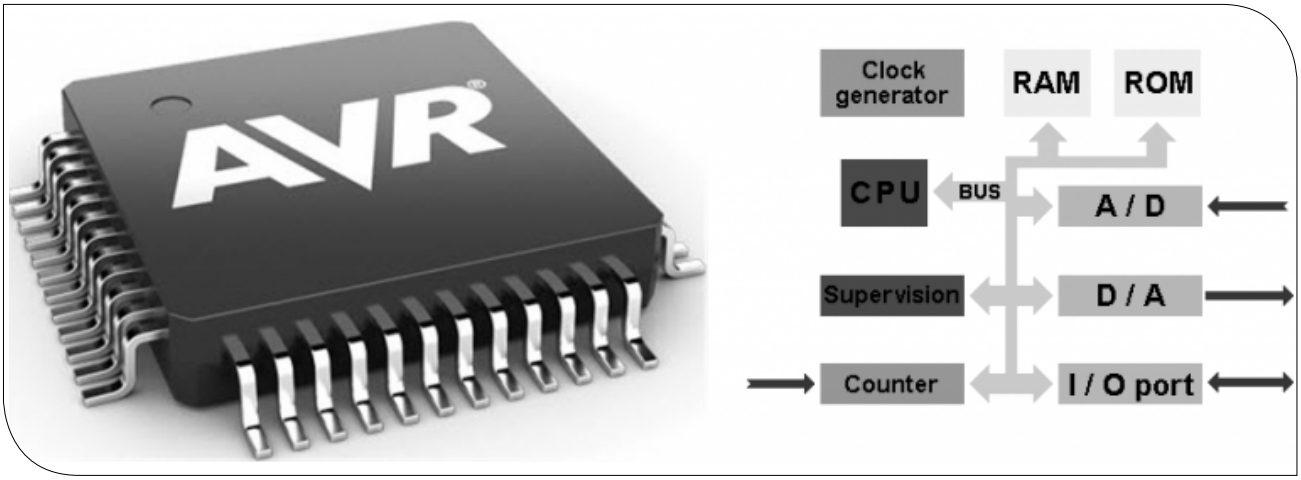



*Figura 1. Microprocessori e loro componenti interni*

| | |
|---|---|
| **Tensione operativa** | 5 V |
| **Pin di I/O digitali** | 14 |
| **Pin digitale di I/O con PWM abilitato** | 6 |
| **Ingressi analogici** | 6 |
| **Corrente DC per ciascun pin di I/O** | 20 mA |
| **Corrente DC per il pin a 3.3 V** | 50 mA |
| **Flash Memory** | 32 kB (ATmega328P) |
| **SRAM** | 2 kB (ATmega328P) |
| **EEPROM** | 1 kB (ATmega328P) |
| **Velocità (Clock)** | 16 MHz |
| **LED_BUILTIN** | 13 |
| **Lunghezza** | 68.6 mm |
| **Larghezza** | 53.4 mm |
| **Peso** | 25 g |

ne è la chiave per l'elaborazione dei dati stessi. Il microprocessore è connesso ad una memoria nonché ad una serie, più o meno numerosa, di dispositivi esterni, i quali servono a fornire dati in ingresso oppure proporre gli stessi in uscita. Questa struttura è tipica di un'unità di elaborazione. Esistono componenti circuitali il cui scopo si esplica spesso all'interno di sistemi integrati per controllo e monitoraggio delle funzionalità di macchinari o di motori. In questo caso parliamo di **microcontrollori** che sono, quindi, **microprocessori dedicati espressamente al controllo** ed è per questo che vengono chiamati microcontrollori. Le grandezze sulle quali sono chiamati ad operare non sono necessariamente solo digitali ma possono essere anche analogiche. Ecco per quale motivo spesso integrano, al loro interno, un multiplexer, un convertitore analogico digitale (ADC), un convertitore digitale analogico (DAC), ed una serie di circuiti di controllo per pilotare attuatori all'uscita. I **DSP**, ovvero **Digital Signal Processor**, sono dispositivi integrati molto simili, dedicati espressamente all'**acquisizione dei segnali** e possono essere impiegati per l'**elaborazione dei segnali** stessi. I DSP sono considerabili come un caso speciale di sistemi integrati che utilizzano microprocessori e che vengono specificatamente indicati soltanto quando c'è da fare elaborazione di segnale. Le differenze non sono soltanto funzionali, anzi, esse derivano da una diversa dotazione hardware. Un microprocessore tipicamente è in grado di comunicare con memorie di vario tipo attraverso l'utilizzo di bus di indirizzi, bus dedicati allo scambio di dati ed anche bus di controllo. Non solo, perché l'interfacciamento avviene tipicamente anche con porte di ingresso e uscita.
L'architettura di un microprocessore prevede al suo interno tre unità fondamentali distinte:

- **ALU**, ovvero l'**Arithmetic Logic Unit**, l'unità centrale che esegue le operazioni aritmetiche.
- **Register Array**, un insieme di elementi di memoria che vengono utilizzati durante l'esecuzione dei programmi per memorizzare dati temporanei oppure per salvare informazioni di rilevanza. Il fatto che la memorizzazione sia temporanea può dipendere dalla specifica applicazione o dall'utilizzo che si sta facendo del microproces-

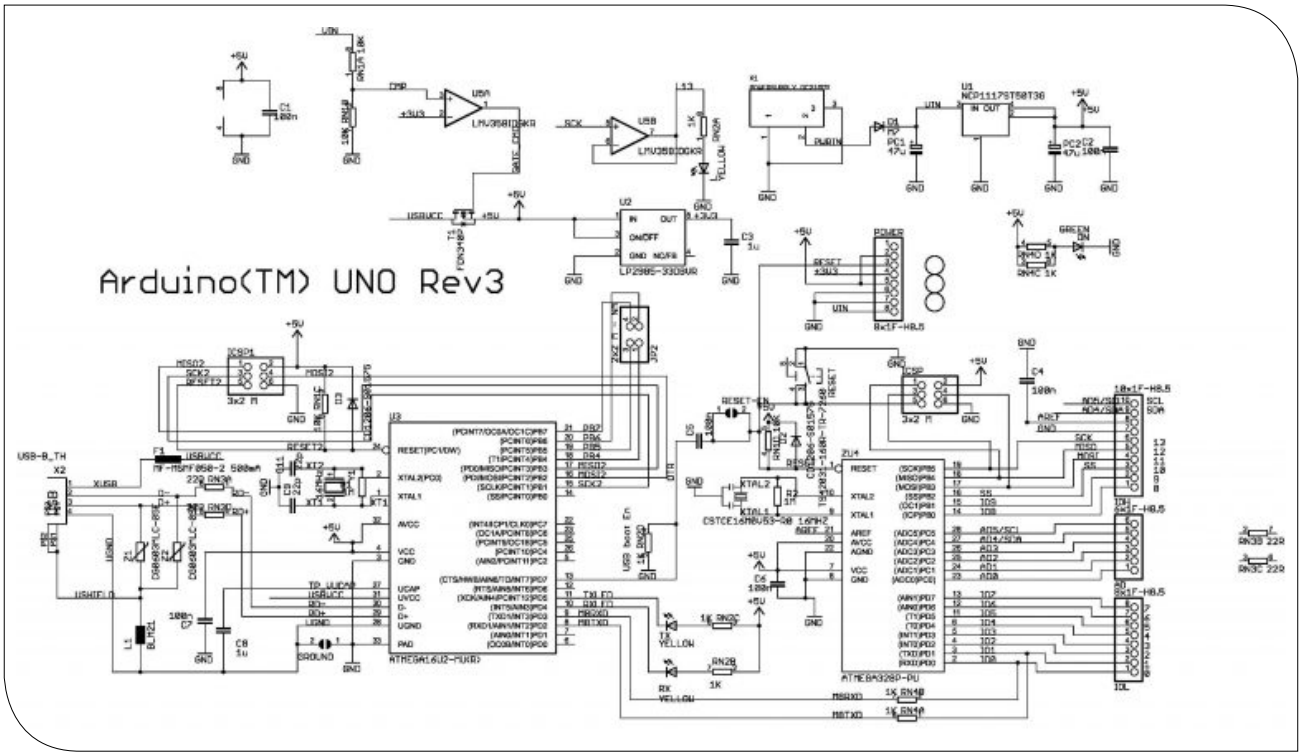



*Figura 2. Schematico della scheda Arduino Uno Rev3*

sore in quello specifico momento.

- **Control unit**, ovvero l'unità deputata al vero e proprio processamento delle istruzioni, alla temporizzazione ed alla gestione dei segnali di controllo per il trasferimento dei dati verso l'interno o verso l'esterno.

## INIZIAMO CON I MICROCONTROLLORI

Cominciamo ad addentrarci nel merito delle **schede per la prototipazione rapida** e focalizziamoci su Arduino, in particolare sulla scheda Arduino Uno. Questa versione della celeberrima scheda è basata su un integrato, l'ATMega328. Si tratta di un microcontrollore a 8 bit che può funzionare con clock a 16 MHz, le cui specifiche tecniche e caratteristiche funzionali sono riassunte nella seguente tabella:

A chiunque utilizzi Arduino come primo approccio alla programmazione ed alla sperimentazione, la scheda, nella sua veste grafica, risulta essere sufficiente. Tutti coloro che, invece, sono un po' più smaliziati o esperti, hanno bisogno di saperne di più. Come sono realizzate le interconnessioni? Qual è lo schema elettrico? In definitiva, quello che si aspettano di vedere è questa rappresentazione:

Di questa rappresentazione sono fondamentalmente importanti tre sezioni, ovvero:

- quella in blu che rappresenta la semplice interconnessione tramite interfaccia seriale emulata e che è accessibile e resa disponibile grazie all'USB;
- la sezione identificata con il verde rappresenta proprio il microcontrollore. Un integrato che si presenta come un componente con dei pin di interconnessione, che espone diverse interfacce, ovvero collegamenti funzionalmente dedicati. La differenza tra un pin e l'altro non è evidente ad una prima ispezione ma lo è nel momento in cui si va a guardare come sono state effettivamente realizzate le interconnessioni. Saper leggere uno schema elettrico è importante perché, quando si realizza un prototipo, bisogna realizzare uno schema che descriva il proprio circuito ed il proprio progetto;
- l'ultima sezione è quella che rappresenta i pin di interconnessione, le sezioni dell'interfaccia. Quella in arancione è la sezione grazie alla quale Arduino riesce ad entrare in comunicazione con il mondo. È importante capire quanti sono questi pin, che funzionalità hanno, e, tra l'altro, anche cosa vuol dire PWM.

Ogni microcontrollore lavora utilizzando un set di istruzioni, ciascuna scritta in linguaggio macchina. Per semplificare il compito della programmazione, come alcuni

di voi sicuramente sapranno, è stato codificato un linguaggio di basso livello noto come assembly, in cui le istruzioni sono date in formato elementare riconducendo tutto a poche semplici istruzioni. Il vantaggio di un linguaggio di alto livello sta prevalentemente nella codifica, ovvero nel fatto che il progettista, il programmatore o lo sviluppatore, possano esprimersi in modo più naturale. Un linguaggio di basso livello è, altresì, più vicino a quello parlato dalla macchina. È importante anche sapere che i microprocessori sono diversi perché possono appartenere a famiglie logiche diverse. Di queste se ne possono individuare due fondamentali, Intel e Motorola. La prima, quella prevalentemente impiegata su tutti i personal computer di tipo IBM, ha alcuni esponenti di spicco che sono:

- 4004
- 8008
- 80x86, da cui i 286, 386, 486, Pentium (I, II, III, IV), Celeron

La famiglia si è evoluta passando dai primi esponenti, che lavoravano soltanto con 4 bit, fino a modelli più avanzati, in uso dal computer che abbiamo quasi tutti nelle nostre case, a 32 o, più frequentemente, a 64 bit. La famiglia Motorola, invece, conta 680X0, da cui 68020, 68030, 68040, 68060. Anche qui, i primi modelli erano ad 8 bit, ma quelli che oggi utilizziamo sono a 64 bit. Alcune altre grandezze importanti nello studio dei microprocessori sono rappresentate dai sistemi di interconnessione, che possono fondamentalmente dividersi in due tipi di collegamento: punto-a-punto oppure multipunto. Come tutti i collegamenti, essi possono essere monodirezionali oppure bidirezionali. Lo scambio di dati è evidentemente facilitato da comunicazioni bidirezionali. Nel funzionamento di un microcontrollore è importante ricordare che il trasferimento di dati può avvenire in serie oppure in parallelo. In teoria, potremmo sempre optare per il trasferimento in parallelo, poichè questo garantirebbe lo scambio di dati in un unico colpo di clock. Quello che succede in realtà è che non è vero che possiamo scambiare N bit in parallelo alla stessa velocità con la quale possiamo farlo per N bit in serie. Lo svantaggio della trasmissione in serie, d'altronde, è rappresentato dal fatto che sicuramente ci vorranno almeno N colpi di clock per trasferire N bit. Quando diciamo che è possibile interfacciarsi con una scheda, dobbiamo sempre capire come lo stiamo facendo. Ecco per quale motivo vediamo nel dettaglio cosa vuol dire avere a che fare con un'interfaccia UART. UART ha lo scopo di trasformare una serie di bytes in un unico flusso, ovvero in un bit stream, il quale viene utilizzato per effettuare le comunicazioni. Quando la trasmissione è in ingresso, il suo scopo è, invece, convertire il flusso effettuando un parsing intelligente per rendere la sequenza di dati comprensibile per il computer. Tra le principali funzioni che svolge vi è il controllo di parità sulle trasmissioni in

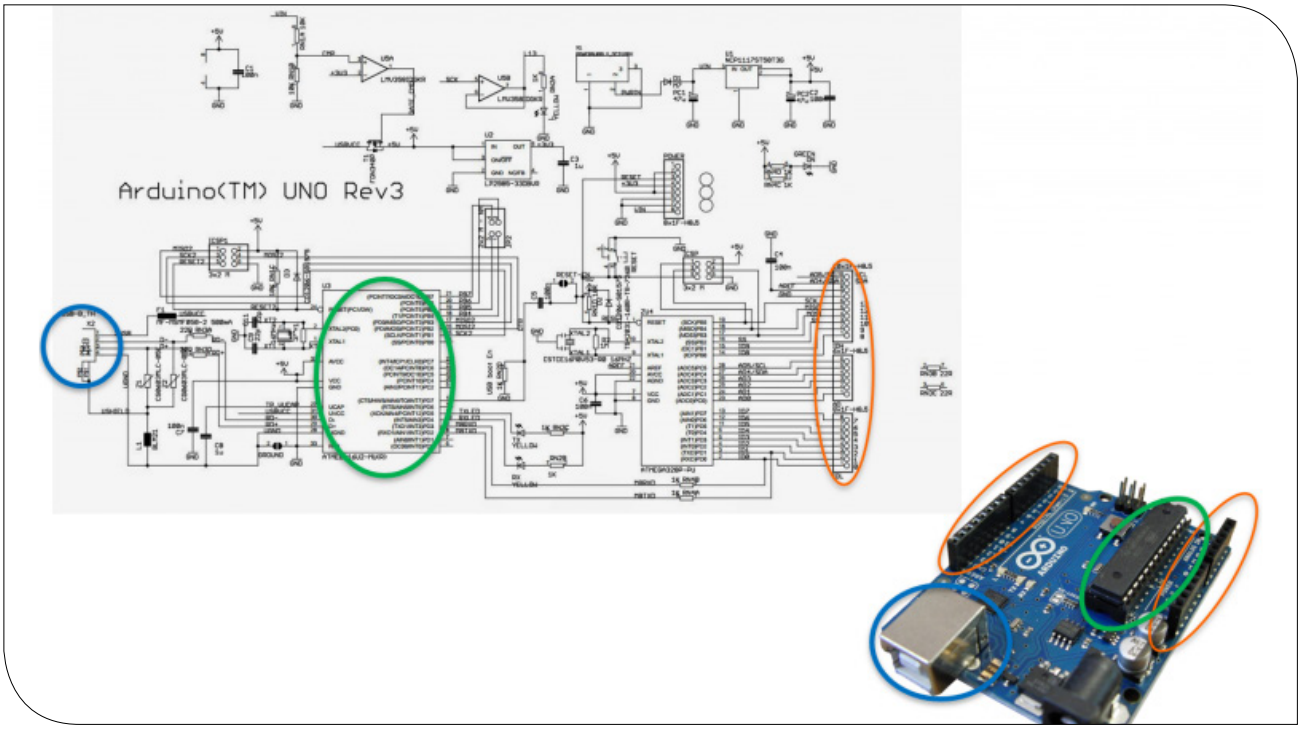



*Figura 3. Dettaglio grafico dello schema di Arduino Uno*

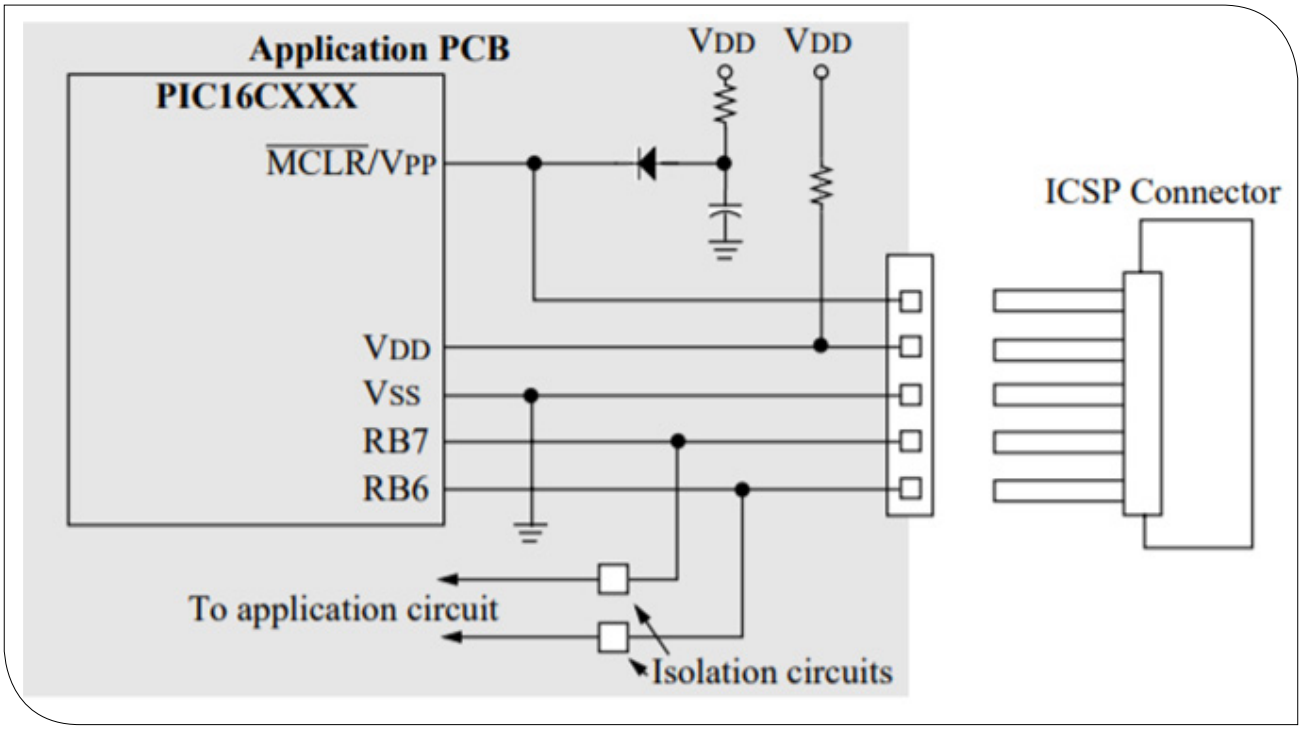



*Figura 4. Schema di collegamento del convertitore ICSP*

uscita. Su quelle in ingresso, in maniera duale, il suo compito è quello di verificare il controllo di parità per validare l'integrità dei dati. Quando si usa il concetto di flusso ordinato, si fa anche riferimento al fatto che si utilizzino dei delimitatori tra pezzi (blocchi) dello stream, in maniera tale da poter effettuare più facilmente una ricostruzione. Questo componente si occupa anche di aggiungere delimitatori di inizio e fine. Dal momento che si tratta di operazioni seriali, è necessario anche comprendere quando esse iniziano e finiscono, perché il flusso di dati altrimenti sarebbe interrotto. Per tale motivo, tramite questo componente vengono gestiti gli interrupt che segnalano l'inizio e la fine delle trasmissioni. Un componente di questo tipo è anche in grado di gestire gli interrupt in funzione del coordinamento delle operazioni nelle comunicazioni tramite computer, specificando anche la quantità di bit al secondo che transitano attraverso l'interfaccia. **Sincronizzare le velocità di trasmissione dati è fondamentale**, come sanno tutti coloro che hanno già provato ad utilizzare Arduino e hanno sperimentato cosa accade quando l'apertura dell'interfaccia seriale avviene impostando un certo baudrate ma poi seleziona una diversa come opzione all'interno del monitor seriale. Con i microcontrollori è necessario sempre comunicare in maniera più diretta possibile. Si cerca, quindi, almeno a livello professionale, di evitare interpreti, piuttosto si preferisce dialogare direttamente con gli integrati. Questo perché le probabilità di errore nella trasmissione dei dati diminuiscono vertiginosamente. Inoltre, diminuiscono anche le latenze. Per poter gestire questo tipo di comunicazioni ci sono interfacce standard che, almeno nel caso di Arduino, sono rappresentate da un **ICSP**, ovvero **In-Circuit Serial Programming**. Esternamente l'interfaccia espone 6 contatti differenti, due linee per la trasmissione dei dati ed il sincronismo e tre dedicate all'alimentazione.

**La programmazione di un microcontrollore avviene in linguaggio C**, quindi non sono, almeno in questo caso, indispensabili le competenze per programmare in assembly. Nello specifico, Arduino propone all'utente un meccanismo estremamente semplificato di interazione con il microcontrollore grazie all'utilizzo della cosiddetta **Arduino IDE, un'interfaccia di programmazione scritta in Java che consente di inviare comandi alla scheda**. Arduino IDE garantisce allo sviluppatore alcune delle funzionalità più gradite a chi scrive software, come ad esempio l'evidenziazione dei costrutti e delle variabili, così come la segnalazione dei tipi, dei metodi e delle funzioni e l'indentazione automatica del codice. È importante, però, sapere che niente di tutto quello che viene programmato può prescindere dalla condivisione. Arduino IDE è il risultato della condivisione, all'interno della community, di diverse modifiche fatte nel corso del tempo. Concordemente, anche l'interazione con il mondo fisico attraverso sensori viene fatta utilizzando delle librerie dedicate che, molto spesso, prima di diventare

ufficiali, o ufficialmente supportate ed integrate all'interno dell'ambiente Arduino, erano state dei semplici prodotti, o meglio progetti, disponibili su Github. Per tutti quelli che cominciano, è importantissimo sapere che **un codice scritto in Arduino contiene almeno due blocchi fondamentali**. Il primo, il cosiddetto setup, è una funzione, un metodo, di tipo void, ovvero che non restituisce niente quando termina, che contiene il blocco di codice che è necessario eseguire all'inizio, nella fase di startup. Quando la scheda si avvia, esegue quello che c'è all'interno del setup una volta sola, dopodiché passa a ciò che è contenuto nel blocco successivo ovvero nel loop. Anche questo è un costrutto che restituisce un tipo void; all'interno del loop vengono inserite tutte le operazioni che la scheda deve effettuare ciclicamente. Significa che una volta che sarà terminata l'esecuzione di ciò che c'è dentro setup, il microcontrollore comincerà ad eseguire in continuazione ciò che c'è scritto all'interno del loop e questo rimarrà vero fino a quando la scheda sarà alimentata. Tanto per spiegare ai nuovi ingegneri, ai nuovi programmatori ed ai nuovi sviluppatori a cosa possa servire tutto questo e come si possa realizzare, immaginate di dover fare il monitoraggio della temperatura. Il vostro termometro digitale si avvia, termina la fase di setup, dopodiché inizia a fare delle letture periodiche. Volete che la temperatura venga tenuta sotto controllo, quindi il fatto che le misure siano periodiche per voi fa parte del progetto, è qualcosa che volete che accada. Una volta effettuata la misura, volete che venga fatta la successiva e così via dicendo. Ma come si può fare realmente questo? Perché esiste il void loop? Dal punto di vista della programmazione, a quale costrutto si fa riferimento? Questa è una domanda davvero importante perché testa la comprensione e la conoscenza che voi avete dei costrutti basilari non soltanto del C ma di qualunque linguaggio di programmazione. **Eseguire una funzione all'infinito, ovvero finché la scheda è alimentata, si può fare utilizzando il costrutto while**. Sapreste rispondere alla domanda: perché non con il for? E una volta che avete risposto, qual è la condizione che deve essere scritta all'interno del while? Comprendere queste funzionalità, se volete questi trucchetti, è un'utile verifica della loro comprensione. Ovviamente, imparare a scegliere la scheda opportuna, quella più giusta, fa parte dell'intelligenza che voi potete esprimere nella progettazione. Per realizzare qualunque tipo di progetto ha senso prendere una scheda dotata del maggior numero di interfacce possibili, del maggior numero di componenti esterni possibile, che dispone del più ampio range di frequenze di funzionamento oppure che può essere alimentata in modi differenti. Ha senso pensare di dotarsi della scheda top di gamma per poter coprire la maggior parte delle esigenze. Ma quando guardate il sito di Arduino, invece, e state pensando ad un progetto specifico, dovete sapere cosa state cercando. Fortunatamente il sito vi aiuta, prima di tutto

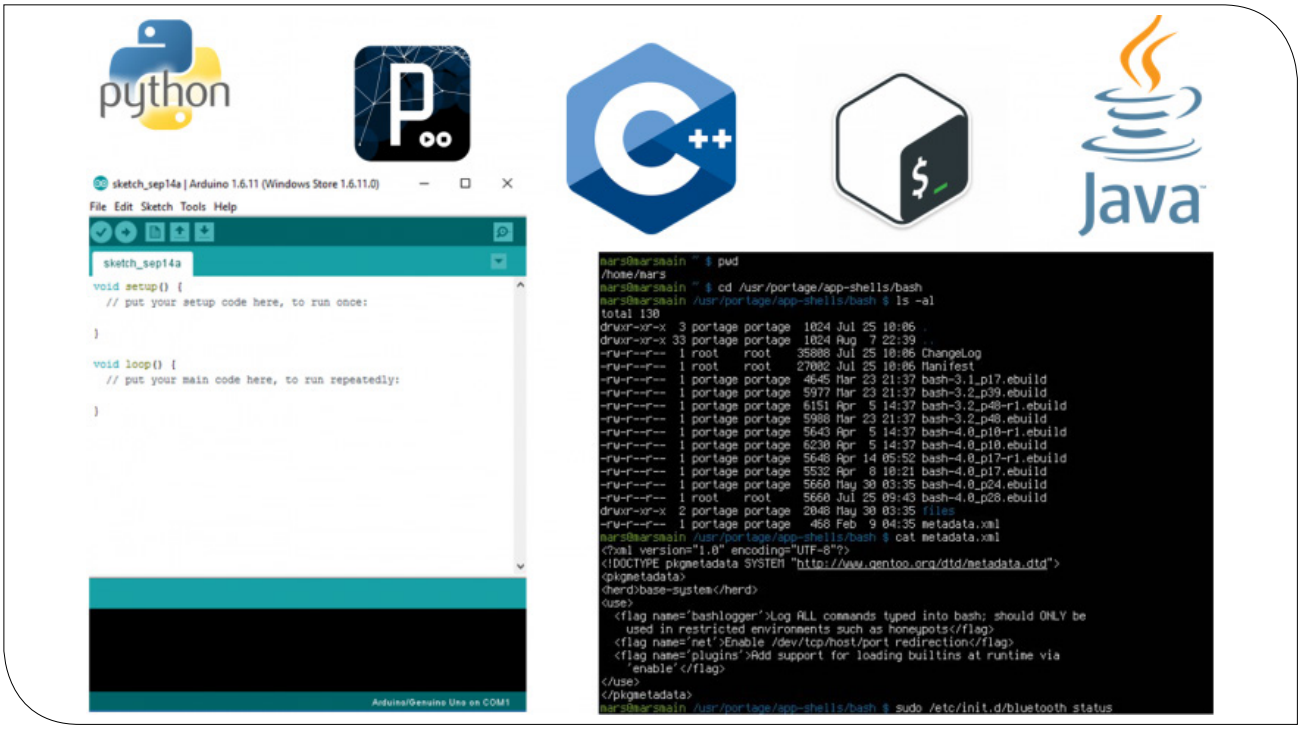



*Figura 5. Vari linguaggi di programmazione utilizzabili con le schede per il rapid prototyping*

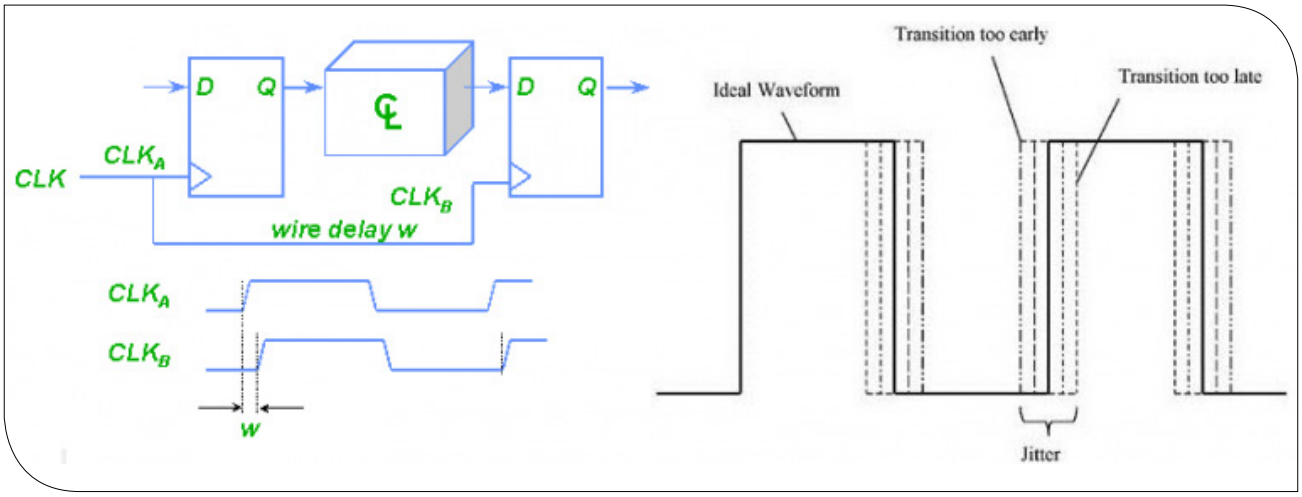



*Figura 6. Fenomeno del jitter sui fronti di salita e discesa dei circuiti digitali*

con un'organizzazione gerarchica secondo categorie funzionali pensate per persone che stanno appena cominciando e che quindi hanno da imparare, come nel caso della categoria entry-level, oppure per chi cerca qualcosa di specifico. Un progetto che vada integrato all'interno di un capo di vestiario, ad esempio, sicuramente ha bisogno di dimensioni e peso molto contenuti ed è proprio per questo motivo che esiste una categoria come quella dei dispositivi wearable.

## COSA CAMBIA CON RASPBERRY PI?

Per provare a semplificare le evidenti differenze che sussistono tra microprocessori e microcontrollori, possiamo riassumere che se gli uni utilizzano un'architettura complessa (CISC, Complex Instruction Set Computer), i secondi utilizzano invece una semplice (RISC, Reduced Instruction Set Computer).

Inoltre è importante ricordare che nei microcontrollori vi sono delle periferiche, come ad esempio I/O, solitamente general purpose, ma anche timers, contatori, ADC ed UART. Molte di queste nei microprocessori non sono implementate. Un microprocessore, inoltre, è dotato di registri di puntamento, registri di indirizzamento e può anche gestire delle code d'istruzione, tutte cose che nei microcontrollori non esistono. Queste non sembrerebbero essere funzionalità necessarie né immediatamente spendibili, ma quando il progetto si fa più complesso diventano molto utili. Nei microcontrollori le memorie RAM e ROM risiedono all'interno del medesimo integrato e pertanto non è necessario utilizzare dei registri dedicati all'indirizzamento. Vi sono anche alcune differenze di ordine pratico: dal momento che i microcontrollori sono realizzati con cMOS (complementary Metal Oxide Semiconductor), la tecnologia consente dimensioni estremamente ridotte. In termini di velocità di funzionamento e processamento dei dati, nonchè esecuzione dei task, i microprocessori sono estremamente più veloci. Come noto, infatti, possono lavorare a frequenze operative dell'ordine dei GHz, ben al di sotto di quanto accade con i microcontrollori che, tipicamente, si fermano ai MHz.

Vedremo più avanti quanto in un progetto IoT il consumo energetico possa essere importante. Bene, considerate sempre che un microcontrollore, solitamente, è ottimizzato per consumare in maniera ridotta, almeno in modalità IDLE, mentre un microprocessore difficilmente ottiene risultati del genere. Al di là del fatto che Raspberry Pi funziona basandosi su microprocessore, è importante ragionare anche su un altro punto. Si propone come un valido supporto alla prototipazione sia dal punto di vista dell'hardware sia dal punto di vista del software perchè, a differenza di quanto accade con Arduino in cui fondamentalmente la scheda è una semplice applicazione di quanto descritto all'interno del datasheet dell'ATMega328, il progetto di un computer per intero, completo e funzionante, il cui schema elettrico è disponibile, rappresenta un prodotto Open Source davvero originale, dall'inizio alla fine. Il suo essere Open Software, però, è diverso da quello di Arduino: se Arduino è un linguaggio di programmazione che si può utilizzare all'interno dell'Arduino IDE, per Raspberry l'essere Open Software significa basarsi prevalentemente su Linux. Il sistema operativo del pinguino, nelle sue varie versioni, infatti, è il vero cuore pulsante della scheda. Poter lavorare con un intero sistema operativo abilita un mondo di possibilità, e ciò che succede con Raspberry Pi è che la scelta della distribuzione più idonea alla propria applicazione può essere fatta in piena libertà dal progettista. Esisto-

no davvero tantissime diverse distribuzioni del sistema operativo che sono ottimizzate per architetture basate su microprocessori ARM. Alcune di queste sono:

- **Raspian**, una distro Debain-based, molto popolare, anche perchè decisamente general purpose. Nelle sue ultime releases è stata notevolmente migliorata dal punto di vista delle performance su architettura ARM anche se in contemporanea è cresciuta la potenza delle schede della famiglia Raspberry soprattutto nella versione 3;
- **Pi MusicBox**, distibuzione specificatamente pensata per trasformare il vostro Raspberry Pi in un vero e proprio jukebox. Basata su Mopidy, la sua ultima relaese funziona benissimo sia su Pi 3 sia su Pi Zero;
- **RetroPie**, una distro specificatamente progettata per tutti gli amanti del gioco da bar. Rivivrete i grandi classici da sala giochi, da Space Invaders a Metal Slug, passando per PAC-MAN;
- **LibreELEC**, una delle tante distro pensate specificatamente per trasformare la vostra TV in una smart TV, il tutto grazie a Kodi;
- **OpenMediaVault (OMV)**, una distro inizialmente pensata per PC, desktop e workstation, OVM consente di realizzare un Network Attached Storage (NAS) dedicato. Niente male se avete molti HDD da connettere.

Una volta presa la scheda, tuttavia, non avete ancora tutto ciò che vi serve. A differenza di quanto accade con Arduino, per utilizzare Raspberry Pi in versione standalone avete sicuramente bisogno almeno di:

- SD card (da almeno 8 GB)
- Display
- Mouse e tastiera
- Power Supply (5 V, da almeno 1 A)

e di alcuni componenti opzionali:

- Cavo ethernet
- USB Wi-Fi dongle (non più necessario nella versione 3)
- cavo audio (non necessario se vi basta l'audio su HDMI)

Per partire con Raspberry non basta collegarlo, come accade invece con Arduino. Se avete scelto Raspian, ad esempio, dovrete:

1. formattare la SD card
2. copiare i file di NOOBS
3. seguire la procedura guidata di installazione
4. effettuare il login

Ci saranno sicuramente degli update o dei tool mancanti, software aggiuntivi che vorrete installare subito. Usate aptitude con, almeno, questi comandi:

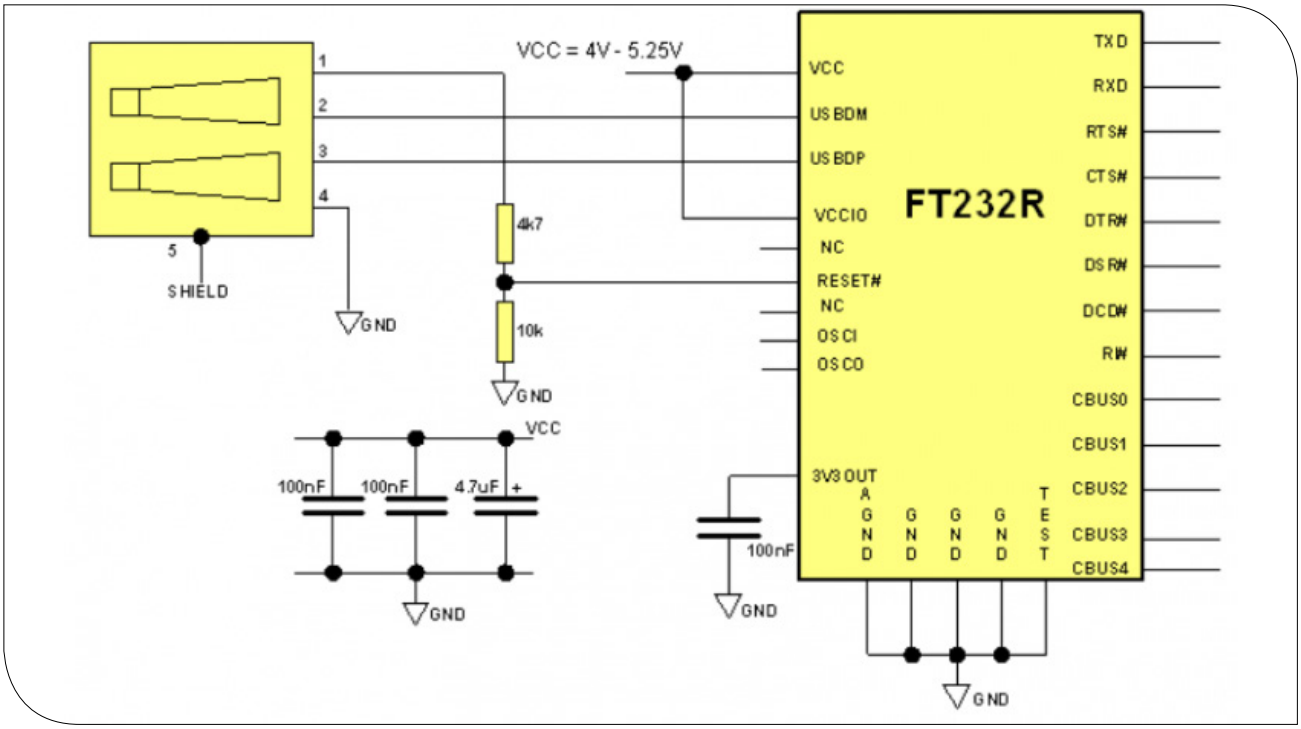



*Figura 7. Rete di connessione per FT232R e USB*

```
# apt-get install /-----/
# apt-get update
# apt-get updgrade
# apt-cache search
```

A questo punto avete installato la distribuzione del sistema operativo scegliendo quella che più vi piace e siete pronti a cominciare. Occorre a questo punto porsi una domanda: con quale linguaggio di programmazione intendo utilizzare Raspberry? Perché si può fare in modalità nativa, utilizzando bash.
Non particolarmente intuitivo, sicuramente un po' scomodo per via della mancanza di un'interfaccia grafica, ma sicuramente essenziale, funzionante e assolutamente preferito dai programmatori e dagli sviluppatori più bravi.
Potreste utilizzare un linguaggio di programmazione di alto livello; su Raspberry Pi quella di utilizzare Python è una scelta molto popolare.
Oppure potreste decidere di utilizzare un altro linguaggio di programmazione.
La scelta dipende esclusivamente da voi.
Anche l'interazione con la scheda richiede che abbiate conoscenze specifiche ma che possono essere garantite da un'intensa attività di pratica. Gestire i GPIO su Raspberry Pi è sicuramente una skill che si può maturare leggendo con attenzione la documentazione.

## INTERFACCE OVUNQUE

Il tema dell'interfacciamento è sicuramente centrale. Come comunico? Cosa? A chi? Come sono fatti i dati che possiedo? In che formato voglio che arrivino? Come garantisco che le informazioni trasmesse/ricevute siano integre? Queste ed altre domande trovano risposte soltanto nella **corretta configurazione dell'interfaccia di comunicazione**.
E questo vale quando si sta spostando un flusso di dati tra un componente ed un altro all'interno di un circuito elettronico ma in egual modo vale nelle reti cellulari e nelle comunicazioni su larga scala.
Vediamo cosa significa progettare l'interfacciamento e quali sono i modi più efficaci per gestire le comunicazioni.
Esistono fondamentalmente due possibilità di comunicare: in maniera seriale oppure in parallelo. I principali parametri da considerare sono:

- **velocità**: in teoria, il data rate di una comunicazione parallela è semplicemente il risultato della

*Figura 8. Fare una misura precisa ed accurata è un po' come giocare a freccette: ci vuole tempo e pratica per riuscirci*

velocità dell'equivalente seriale di ciascuna linea per il numero delle linee. In pratica questo non è vero. Un fattore limitante della velocità nelle comunicazioni in parallelo dipende dalla velocità di ciascun link che, per fenomeni di clock skew, fisicamente inevitabili, implicano un rallentamento;

- **lunghezza**: la lunghezza di un collegamento ha dimensioni ridotte su scheda, ma anche potenzialmente dei difetti, sopratutto in termini di contiguità con altre piste conduttive. Il fenomeno del crosstalk, infatti, ovvero la possibilità che i segnali dati all'interno di una pista possano interferire e trasferirsi, con conseguente perdita di potenza, alle linee continue, rappresenta un fattore che suggerisce che le piste conduttive debbano essere sempre il più possibile di dimensioni contenute e distanti tra loro;
- **complessità**: si tratta di un aspetto fin troppo sottovalutato, soprattutto quando ci rivolgiamo all'utilizzo degli integrati ed alla loro comunicazione. Le comunicazioni in parallelo possono essere la soluzione ai problemi di complessità, di numerosità dei componenti e dei loro collegamenti, nonché delle loro interconnessioni. Questo, però, introduce un tema importante di sincronizzazione e di gestione delle conversioni tra comunicazioni seriale e parallelo e viceversa.

Il problema del clock skew ha un'influenza notevole poichè la precisione dei fronti di salita e discesa di un segnale digitale non è infinitamente grande quanto la modellazione suggerisce. I fronti, infatti, non sono rette a pendenza infinita, ovvero non sono perfettamente verticali.
Questo vuol dire che esiste un certo tempo di commutazione tra il livello basso e quello alto. In questi intervalli temporali è necessario monitorare costantemente l'uscita dei sistemi digitali, onde evitare di fraintendere il transitorio per un simbolo.
La cosa diventa tanto più importante quanto si impiegano politiche di duty cycling all'interno del nostro circuito.

## UART E USART

Sembrano solo due sigle, anche molto simili, ma in realtà tra loro ci sono grandissime differenze e quando ci si deve interfacciare è importante comprenderle.
**UART è un dispositivo per la comunicazione asincrona** in cui il formato dei dati e le velocità di trasmissione sono configurabili. I livelli e i metodi del segnale sono gestiti da un circuito driver esterno.
Come accennato in precedenza, **le UART sono comunemente incluse nei microcontrollori**. Il loro utilizzo si basa su un segnale di clock interno, grazie al quale i dati sul bus possono avere un timing regolare.
La gestione di questa interfaccia richiede che vi sia non soltanto una temporizzazione ma anche bit dedicati che segnalino avvio ed arresto di una sequenza, in maniera tale da ottenere sincronizzazione dei dati asincroni. Un **trasmettitore ricevitore sincrono asincrono universale (USART)** può agire in modalità asincrona proprio come un UART ma ha anche il vantaggio, ovvero la capacità, di operare in modo sincrono. I dati sono sincronizzati da un clock il quale può essere:

- recuperato dai dati
- inviato come segnale esterno

A differenza di UART, USART non utilizza bit di inizio e fine. Questo è uno dei fattori che incide sul fatto che USART abbia un baud rate più alto. Vediamo nel dettaglio FTDI FT232R, un'interfaccia UART per dispositivi USB, proprio come nel caso di Arduino.

## IL RESISTORE COME INTERFACCIA

Anche un semplice resistore può essere considerato un'interfaccia. Nella sua definizione, infatti, un'interfaccia non è altro che una soluzione che consente la traduzione di una grandezza in un'altra ovvero l'utilizzo di qualcosa a mezzo di uno strumento.
Un resistore, ad esempio, consente la conversione tra un valore di tensione ed uno di corrente e viceversa. Si tratta di un sistema che ha una funzione di trasferimento, e come tutti i sistemi reali, esso stabilisce una diretta proporzionalità tra tensione e corrente per cui il valore della resistenza cambierà.

## IL WI-FI

Allo stesso modo anche una scheda di rete che consente l'interconnessione del proprio dispositivo con altri connessi ad una rete, può essere considerata un'interfaccia.
Come tutte le interfacce anche una scheda Wi-Fi ha delle specifiche tecniche:

- data transfer rates (misurato in Mbit/s), che va da 2 Mbit/s fino anche a 54 Mbit/s
- potenza in trasmissione (misurata in dBm)
- range di copertura (misurato in m), che si estende entro 100 m
- supporto per diverse versioni dello standard (ovvero, 802.11 b/g/n)

| Variabile fisica | Principio di rilevazione |
|---|---|
| Distanza/posizione | Magnetico |
| | Elettromagnetico |
| | Effetto Hall |
| | Optoelettronico |
| Accelerazione | Elettrostatico |
| | Elettromagnetico |
| | Effetto Hall |
| Pressione | Elettromagnetico |
| | Piezoelettrico |
| Temperatura | Semiconduttori |
| | Termocoppia |
| Visione | Optoelettronico |

e altrettanto, come tutte le interfacce, richiede degli elementi per la sua configurazione essenziale:

- Indirizzo IP per l'interfaccia Wi-Fi (statico o dinamico)
- Service Set IDentifier (SSID)
- Autenticazione per abilitazione del router
- Configurazione di un passphrase se si sceglie WPA/WPA2

## GLI ELEMENTI SENSIBILI

Discorso del tutto analogo e che si allaccia perfettamente a quello sulle interfacce, è ciò che possiamo fare sul principio fisico di rilevamento di una grandezza di nostro interesse.

Esattamente come quando diciamo di voler monitorare la temperatura, dobbiamo specificare se stiamo parlando della temperatura di esercizio di un motore, piuttosto che quella di fusione di un dato materiale o ancora quella di un altoforno, in tutti i campi della tecnica è indispensabile specificare le condizioni di esercizio. La temperatura corporea ha senso misurarla in un range che non può essere ovviamente inferiore a 40 °C e sicuramente non sarà mai superiore a 100, almeno per quanto riguarda le condizioni compatibili con la vita e quelle immediatamente prossime alle cause di morte. Se invece vogliamo creare un termometro per verificare se una persona abbia o meno la febbre, il suo range operativo si restringe ulteriormente perché non ha senso andare sotto i 34 °C e sicuramente non è più compatibile con la vita una temperatura superiore ai 45/46 °C. A seconda, quindi, di ciò che vogliamo misurare e delle condizioni di lavoro, il nostro principio di misura può essere lo stesso ma lo strumento potrà cambiare. Altrettanto si può dire per un termometro da cucina, in cui per verificare ad esempio la corretta temperatura di ebollizione dell'acqua, il nostro termometro dovrà essere in grado di arrivare a 100 °C senza che questo ne causi la rottura o ne impedisca il funzionamento.

Possono sembrare delle considerazioni abbastanza banali ma in realtà il progettista deve fare questo lavoro, ovvero identificare con estrema precisione tutte e sole le condizioni operative. Supponiamo di voler monitorare una struttura, ad esempio un ponte. Un ponte ha determinate caratteristiche, ha una propria lunghezza così come una determinata ampiezza dell'arcata, alcuni elementi strutturali portanti, altri che sono soltanto di supporto e così via dicendo.

Per poter misurare la tenuta del ponte, la sua operatività, il fatto che la sua posizione sia corretta e che non subisca deformazione o flessioni o torsioni, è importantissimo capire come misurare queste grandezze. Quali tipi di sensori bisogna utilizzare? Quali sono i posti più corretti per posizionare gli elementi sensibili? Quanti elementi sensibili bisogna utilizzare? Queste sono alcu-

ne delle domande alle quali il progettista deve rispondere per poter correttamente configurare lo studio preliminare del proprio sistema.
Gli elementi sensibili in questo esempio non sono altro che degli estensimetri che verificano eventuali sforzi cui la struttura è sottoposta.
A ciò si affianca l'elaborazione compiuta da parte del sistema per poter garantire un esatto controllo delle vibrazioni.
Tramite analisi mirate è possibile, peraltro, verificare che le vibrazioni alle quali il ponte è soggetto durante l'esercizio siano entro i limiti consentiti, quelli che garantiscono che la struttura è ancora in sicurezza. Facciamo ancora un'altra ipotesi e immaginiamo di voler progettare un sistema di illuminazione per interni.
La prima cosa che bisogna assolutamente comprendere è la destinazione d'uso.
È un interno ad uso abitativo? Una casa? Un ufficio? Un luogo di lavoro? Un locale commerciale? Un museo?
Queste domande sembrano banali, ancora una volta, ma non lo sono.
All'interno di un ambiente domestico, ad esempio, per poter garantire il comfort abitativo, è giusto pensare di utilizzare luci calde che tendono al giallo oppure all'arancione o ancora al rosso.
In un ambiente di lavoro in cui si cerca di simulare l'illuminazione a giorno, la luce deve essere bianca, fredda, il più possibile chiara, deve illuminare tutto l'ambiente che, possibilmente, deve avere anche armadi, strutture ed elementi di arredo che tendano anche essi a colori chiari. In un locale commerciale viene preferita l'illuminazione diffusa, in maniera tale da poter dare ai clienti l'esperienza utente più completa nella fruizione degli scaffali che espongono le merci.
Quando si illumina un museo, invece, l'illuminazione di solito è da un lato diffusa, ma non troppo intensa, dall'altro puntuale in maniera tale da poter esaltare le singole opere esposte, piuttosto che le sculture. Ancora una volta stiamo progettando soltanto luci ma naturalmente la scelta della luce cambia completamente. Poi esistono alcune domande che dobbiamo porci dal punto di vista tecnologico.
Ad esempio: vogliamo utilizzare delle luci a incandescenza, possiamo pensare di utilizzare l'illuminazione a LED, la luce deve essere estremamente monocromatica? È ovvio che queste domande non sono solo inerenti il tipo di lampadina piuttosto che punto luce da utilizzare.
La scelta della tecnologia impatta moltissimo sui costi, sull'illuminazione che la plafoniera è in grado di diffondere, sulla manutenibilità ovvero su qual è il range di operatività e il tempo di vita misurato in ore.
Una volta che è stata scelta l'impostazione tecnologica del progetto, è bene considerare che le variabili in oggetto possono essere fondamentalmente di due tipi, ovvero grandezze continue o grandezze discrete.
La temperatura è tipologicamente una variabile discreta, dal momento che varia con continuità ma in un intervallo discreto.
La temperatura corporea difficilmente viene espressa con precisione oltre la seconda cifra decimale, pertanto si tende a considerare che 37,24 gradi centigradi sia una temperatura corporea ma difficilmente la misureremo arrivando oltre, ovvero non diremo mai 37,24 °C.
Diversamente viene fatto con il tempo che di per sé è una variabile continua descrivibile tramite numeri reali. In realtà quando la descriviamo attraverso un calcolatore oppure utilizziamo un cronometro, per poterla misurare ci fermiamo al millesimo di secondo, eventualmente aumentando il numero di cifre significative dopo la virgola ma comunque facendola diventare una variabile discreta.
**La differenziazione tra variabile continua e variabile discreta ha un impatto sul tipo di sistema che andiamo a considerare**.
In un circuito digitale tutte le variabili sono sicuramente discrete nella loro definizione ed assumono quindi valori discreti.
Questo significa che dobbiamo fare un'altra distinzione, ovvero le grandezze possono essere continue nelle ampiezze oppure assumere valori discreti.
Tipicamente il secondo caso si verifica quando le variabili vengono passate all'interno di un sistema di elaborazione di dati per cui il dato viene espresso utilizzando un certo numero di bit, 8, 10, 12 o 24 che siano.
Di seguito riassumiamo alcune delle caratteristiche principali dei sensori:

- accuratezza
- precisione
- risoluzione
- sensibilità
- caratteristica statica
- caratteristica dinamica
- intervalli di input e output
- isteresi, linearità, deriva, impedenza di uscita, soglia
- principio di misurazione
- unità di misura

Nella tabella che segue vengono rappresentati alcuni principi fisici di rilevazione e, concordemente, alcune ti-

pologie di sensori adatti alla misurazione di queste stesse variazioni:

## AVVERTENZE PER UN USO CORRETTO DELL'IOT

Detti questi principi generali, prima di avviarci alla conclusione di questo articolo e iniziare a mettere le mani su progetti che hanno realmente una caratteristica smart e che quindi, a pieno titolo, rientrano nell'ambito IoT, è opportuno fare alcune importanti precisazioni:

- La prima caratteristica dei dispositivi e dei sistemi IoT è che le risorse sono molto limitate. Questi dispositivi sono Smart per via delle funzioni che svolgono ma l'intelligenza si trova all'interno di dispositivi con capacità di calcolo estremamente ridotta, come avviene per i microcontrollori ad 8 bit e che funzionano ad una frequenza, ad esempio, tipica di 8 MHz. Concordemente sono estremamente risicati nella loro dotazione di memoria, in termini di RAM e ROM. Sono anche estremamente ridotti in termini di alimentazione, vengono tipicamente alimentati a batteria, con batterie molto piccole e che hanno bisogno di essere spesso sostituite se l'utilizzo del dispositivo diventa frequente. Ecco per quale motivo **una delle principali sfide nei dispositivi IoT è l'ottimizzazione dei consumi energetici**.
- Un'altra caratteristica importantissima dei dispositivi nell'era dell'Internet of Things è che **le applicazioni sono distribuite su più nodi**. I sistemi IoT sono tipicamente auto organizzati, con **intelligenza decentralizzata e distribuita**, con nodi che sono in grado di svolgere i loro compiti in perfetta autonomia. Inoltre, i cosiddetti end devices, ovvero i nodi periferici di queste ipotetiche topologie, che molto spesso cominciano a somigliare a delle mesh complete, sono in grado di coordinarsi tra loro nella operatività e nell'esecuzione dei loro compiti ma il loro comportamento è fortemente influenzato dalle variabili d'ambiente: se stiamo, ad esempio, pensando a sistemi di monitoraggio di una catena di produzione o di una linea industriale, il sistema controllerà lo stato dei macchinari e ogni volta notificherà se ci sono anomalie funzionali, specifiche non rispettate, temperature di esercizio inusuali e così via dicendo.
- Molto spesso nei sistemi caratterizzanti l'IoT, l'esecuzione dei task avviene per un breve istante di tempo oppure in un periodo comunque ristretto. È necessario gestire l'acquisizione e l'elaborazione dei dati dai sensori, la trasmissione e la ricezione degli stessi attraverso le interfacce di comunicazione in breve tempo. Utilizzando questa metodologia operativa si ottimizzano i consumi energetici esattamente come accennato in precedenza.
- In applicazioni particolari, come ad esempio in ambito industriale, i sensori, i sistemi ed i dispositivi IoT devono essere robusti e resistenti anche all'operatività in ambienti ostili e/o che propongono condizioni estreme di funzionamento, come temperature molto alte oppure elevati livelli di acidità.
- Ultima caratteristica fondamentale deriva un po' dal raccordo di tutto ciò che è stato detto fino a questo momento. Quando si chiede a un dispositivo fisico di poter fare sia rilevazione e condizionamento del segnale che una prima elaborazione e nel contempo anche creare una struttura dati per poterle inviare in maniera ottimizzata, si sta facendo formalmente un po' confusione tra il ruolo dell'hardware e quello del software. Pur tuttavia, una caratteristica fondamentale dei dispositivi IoT è proprio **l'assenza di una distinzione netta tra ciò che è hardware e ciò che è software**, ciò che è programmazione e ciò che invece è scheda, dispositivo, componente, circuito. Il sistema IoT risulta essere tale soltanto se la fusione di tutti questi elementi insieme garantisce funzionalità avanzate.

## CONCLUSIONI

Siamo giunti alla fine di questo appuntamento con la prototipazione rapida.

Oggi abbiamo visto nel dettaglio cosa significa avere a che fare con degli strumenti, studiarli per comprenderli nel dettaglio, utilizzarli in maniera opportuna ed interconnetterli in maniera corretta.

Dalla prossima volta proveremo a mettere in pratica queste considerazioni e a trasformare gli oggetti di uso comune in veri e propri dispositivi IoT rendendoli davvero smart.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/iot-e-chi-iot-lo-fa-comprendiamo-gli-strumenti

# IOT È CHI IOT LO FA: IL PORTASPICCIOLI SMART

di **Pietro Boccadoro**

*Negli articoli precedenti abbiamo approfondito cosa significa creare un prototipo, progettandolo fin dalle basi. Ma dopo aver seguito un approccio teorico è giusto provare a declinare tutto ciò in maniera pratica. Cosa significa quando abbiamo un'idea di progetto, declinare tutte le specifiche tecniche funzionali? Come si fa a stimare il microcontrollore adatto oppure decidere quanti sono i sensori che dobbiamo collegare? In questo articolo ci focalizzeremo su come creare un vero e proprio progetto IoT. In particolare, quello che faremo sarà rendere l'IoT un oggetto di uso comune: un semplice portaspiccioli. Siete pronti?*

## INTRODUZIONE

In questo articolo vedremo un esempio concreto di **prototipazione rapida**, una **tecnologia altamente innovativa che consente di produrre oggetti di geometria anche complessa e con determinate features, in tempi molto ridotti, partendo dalla definizione matematica dell'oggetto realizzato utilizzando un modello CAD tridimensionale**. Gli oggetti vengono ottenuti con progressiva aggiunta di materiale. Inoltre, l'idea di base di questa serie di articoli è quella di provare a declinare l'Internet of Things nella realtà di tutti i giorni, che viviamo quotidianamente. Il punto di partenza dell'Internet delle Cose è che gli oggetti che vediamo e utilizziamo ogni giorno possono diventare intelligenti, in sostanza smart. Una penna, ad esempio, è un oggetto comune che di base permette di scrivere, ma una penna che fosse in grado anche di comunicare ad un computer ciò che è stato scritto su un supporto cartaceo, sarebbe

*Figura 1. Moleskine smart al lavoro (Smart Writing System)*

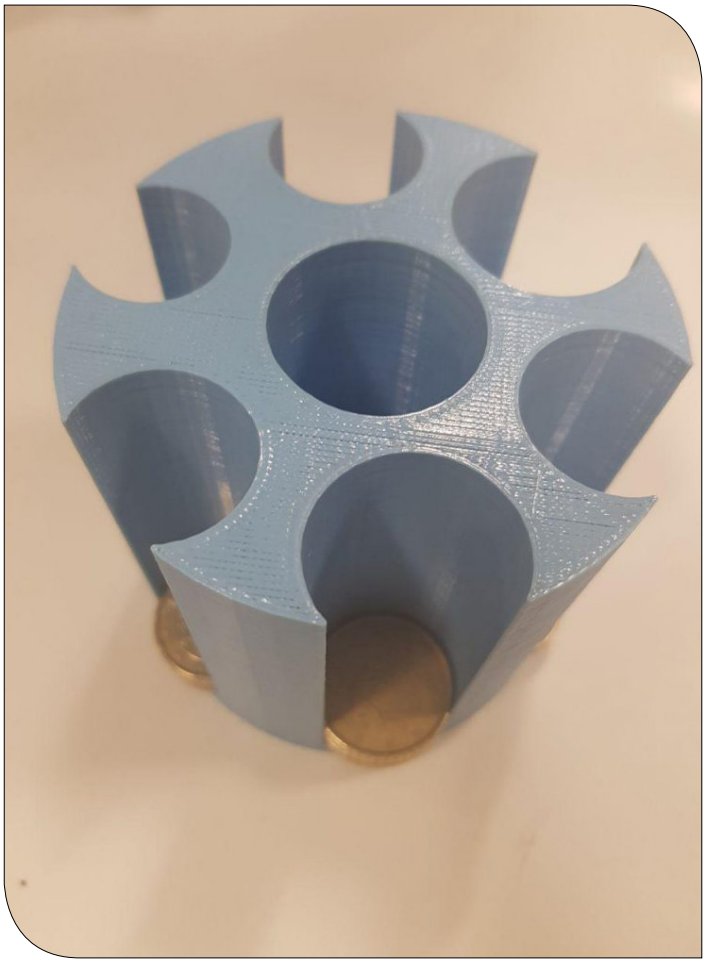


*Figura 2. Portaspiccioli "analogico" stampato in 3D*

già molto più smart. Qualcosa del genere è stato realizzato ad esempio dal noto brand Moleskine che ha trasformato il celeberrimo blocchetto di appunti utilizzato da alcuni tra i più grandi artisti di sempre, in qualcosa di digitale e altamente innovativo, decisamente più al passo con i tempi, lo Smart Writing System.

Questo è solo uno dei tanti esempi di come una grande azienda prova a convertirsi e a innovare i propri prodotti. Noi siamo makers, hobbisti, progettisti, ingegneri o comunque persone molto creative ed il nostro scopo è quello di divertirci a trasformare gli oggetti che comunemente troviamo in casa. Vediamo come la progettazione basata sulla prototipazione rapida può aiutarci in tutto questo. Supponiamo di avere tra le mani un oggetto di uso comune, decisamente analogico, un semplice portaspiccioli. All'interno di un portaspiccioli solitamente depositiamo tutti gli spiccioli che abbiamo, ci liberiamo le tasche e molto spesso non siamo interessati a sapere quanti soldi si stanno accumulando. Ciò nondimeno, nel corso del tempo, quelli che prima erano fastidiosi pezzi di metallo, potrebbero costituire una vera e propria fortuna. E allora perché non tenerne traccia? Quello che vedete nell'immagine che segue è un esempio di portaspiccioli realizzato con una

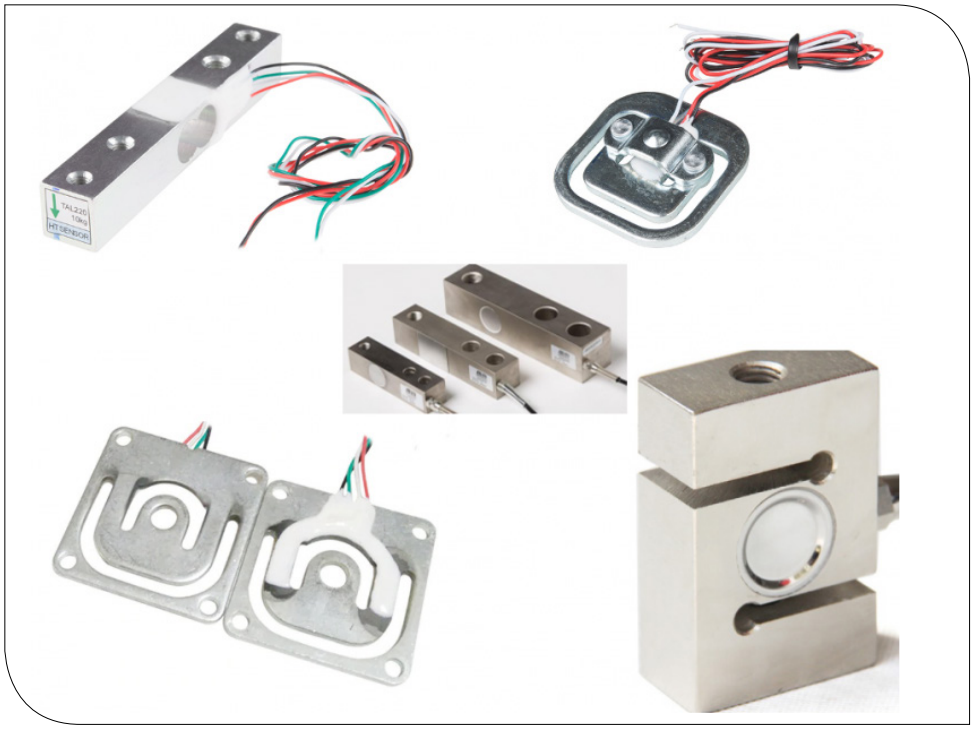


*Figura 3. Panoramica delle varie celle di carico*

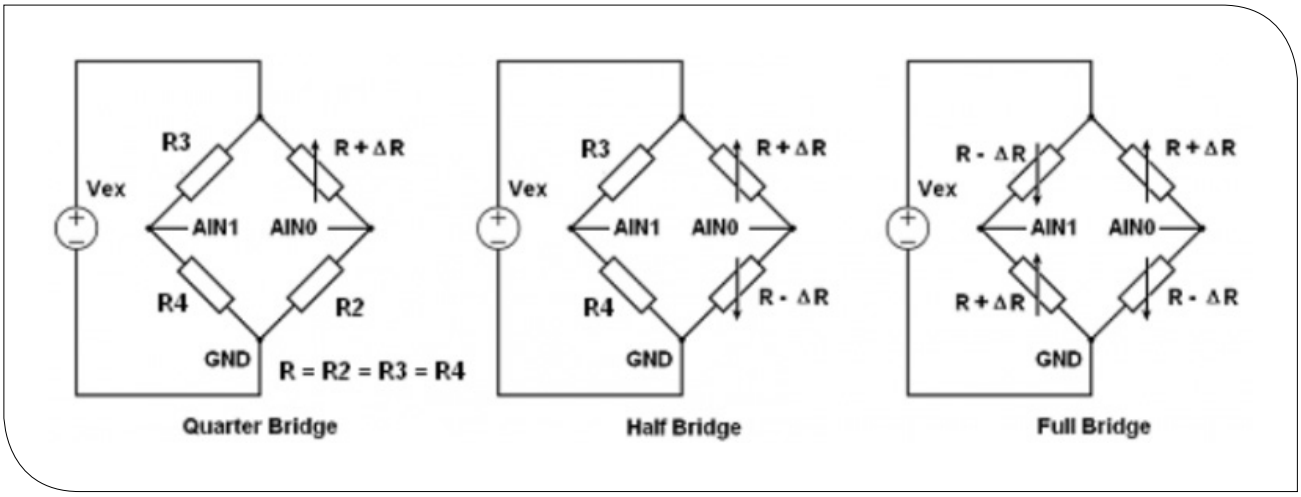



*Figura 4. Differenti configurazioni (quarto di ponte, mezzo ponte, ponte completo) del ponte di misura gestito dallo Strain Application Adaptor EMANT300.*

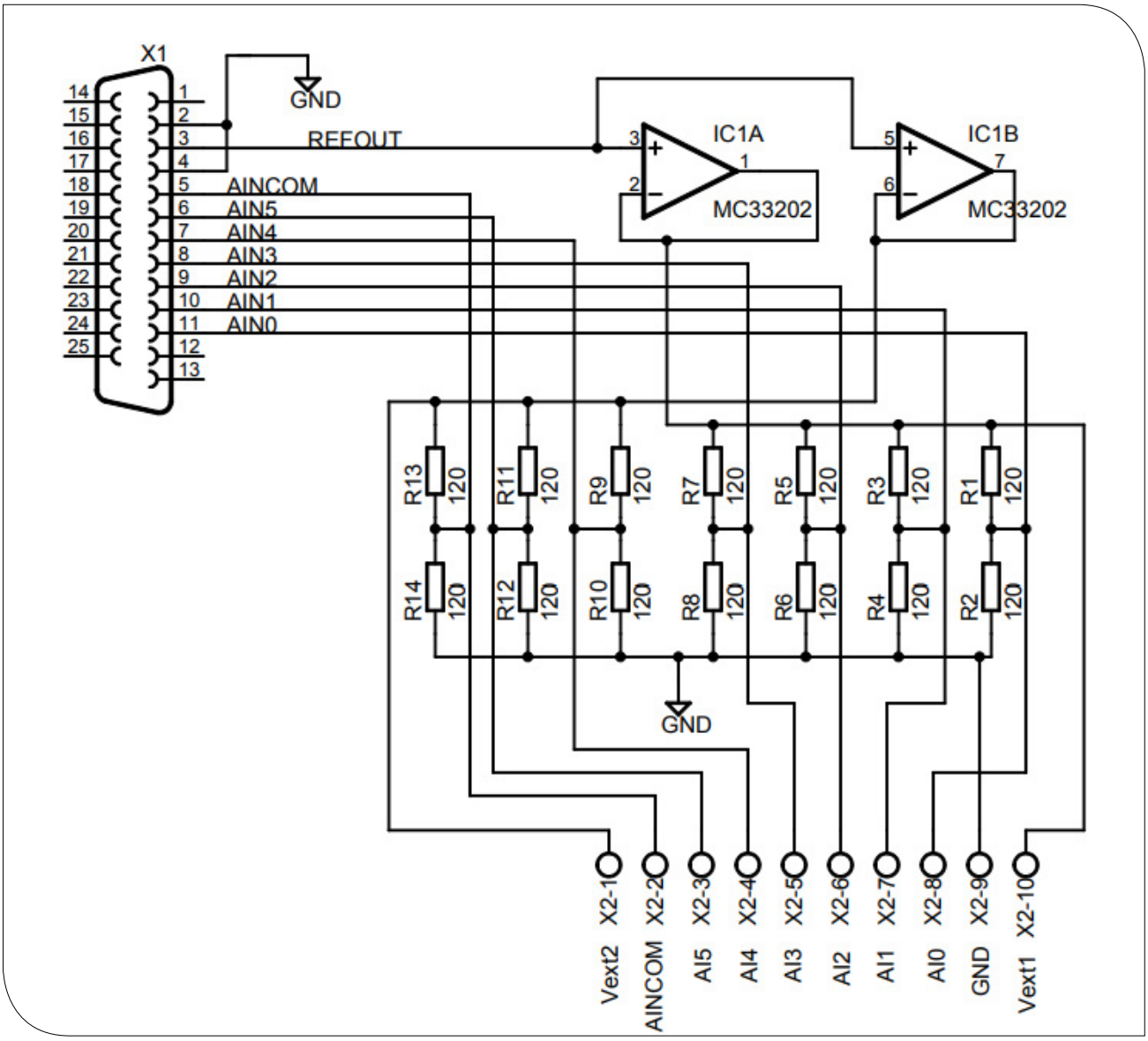



*Figura 5. Schematico dell' EMANT300*

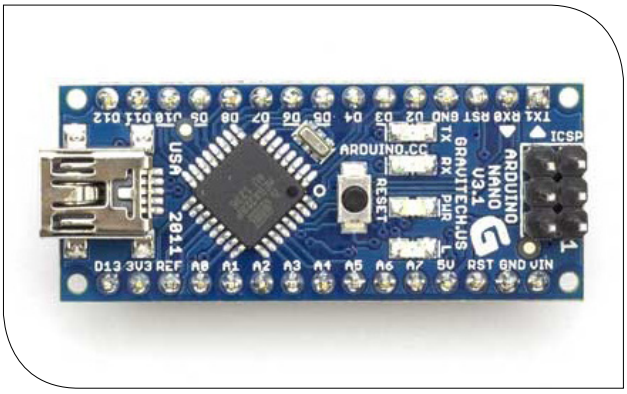



*Figura 6. Scheda Arduino Nano*

stampante 3D.

La domanda da porsi a questo punto è: come faccio a sapere esattamente quanti soldi ci sono all'interno di questo oggetto? Per rispondere dobbiamo cercare di comprendere esattamente che cosa fa di base questo manufatto e poi provare a intervenire sul suo funzionamento.

Guardandolo ci rendiamo conto che non si tratta semplicemente di un banale borsellino, all'interno del quale gli spiccioli vengono inseriti in maniera confusa e non distinguibile.

Questo non è un semplice portaspiccioli, ma è un vero e proprio organizer, ovvero gli spiccioli sono contenuti all'interno di vani dedicati, che dividono le monete identificando le loro differenti dimensioni.

Questo significa che possiamo guardare lo spicciolo come un oggetto che ha determinate proprietà. E in effetti noi sappiamo che tutti gli spiccioli hanno diverse caratteristiche (features) quali:

- Zigrinatura
- Dimensione
- Peso
- Materiale
- Colore

È intuitivo, quindi, che per poter rendere intelligente questo oggetto, dovremmo portare alla luce e rendere il più possibile evidenti queste features.

Una volta fatto ciò, dovremmo poter essere in grado di sfruttare tali caratteristiche per poter differenziare le monete ed in definitiva poterle contare.

A questo punto del nostro processo la sfida è chiara: dobbiamo poter distinguere quante sono e quali sono le monete presenti all'interno di ciascuno degli scomparti di questo oggetto.

Per farlo è necessario, anzi indispensabile, comprendere esattamente **quale grandezza del mondo fisico può essere misurata per poter identificare quello che stiamo cercando**, ovvero una differenza di dimensioni piuttosto che una differenza di peso.

In pratica, stiamo cercando il **principio fisico di rilevazione**, ovvero **la variabile che varia e che noi possiamo misurare**.

Una volta che avremo capito qual è, passeremo al principio di misurazione, che diventerà il nostro oggetto di studio.

Pensando a questo progetto ci viene in mente che non siamo i primi a voler stimare se una moneta sia di un taglio piuttosto che di un altro. Tutti i distributori automatici, infatti, effettuano questa operazione.

Cosa c'è di diverso, allora, tra ciò che vogliamo fare noi e ciò che viene fatto all'interno dei comuni distributori automatici? Come funziona esattamente questo macchinario? Quando la moneta viene inserita, vengono controllati contemporaneamente diametro, peso, e materiale.

Dopodiché la moneta viene inserita all'interno di una cassa comune, il deposito è unico e non tiene conto delle differenze tra le monete. Il nostro portaspiccioli invece è diverso poichè abbiamo già fatto la selezione delle monete e abbiamo già identificato in quale compartimento devono essere inserite.

L'inserimento è analogico, ovvero siamo noi che inseriamo la moneta all'interno del portaspiccioli e questo significa che non siamo noi ad essere l'intelligenza digitale del sistema. Come facciamo, allora, ciò che vogliamo realizzare? Come rendiamo intelligente e smart questo oggetto?

Ragioniamo inizialmente su una possibilità: il peso. Il peso del portaspiccioli in sè può rappresentare la nostra tara, ovvero un offset.

Come si effettua la misurazione del peso? Si può fare con una molla, una dinamo oppure con una **cella di carico**.

Bene, abbiamo capito che esiste un modo per rilevare il peso ed abbiamo anche un elemento sensibile.

Come leggiamo il valore che viene misurato? Per effettuare la misurazione è necessario disporre di un circuito di condizionamento ed un sistema di misurazione a ponte che sia in grado di effettuare la lettura del dato.

Dopodiché, anche in questo caso, come tutti i sistemi, abbiamo delle specifiche tecniche come ad esempio lo schema elettrico dell'integrato.

Qual è il prossimo step? E' presto detto, si studiano le interfacce.

Come posso comunicare con questo oggetto?

Le interfacce esposte possono essere di tipo seriale o

| | |
|---|---|
| **Microcontrollore** | ATmega328 |
| **Architettura** | AVR |
| **Tensione di lavoro** | 5 V |
| **Memoria Flash** | 32 kB |
| **SRAM** | 2 kB |
| **Frequenza di clock** | 16 MHz |
| **Pin di Input Analogico** | 8 |
| **EEPROM** | 1 kB |
| **Corrente DC per ciascun pin di I/O** | 40 mA (I/O Pins) |
| **Tensione di ingresso** | 7-12 V |
| **Pin di I/O digitali** | 22 (di cui 6 con PWM) |
| **PWM Output** | 6 |
| **Consumo di corrente** | 19 mA |
| **Dimensioni** | 18 x 45 mm |
| **Peso** | 7 g |

parallelo e possono effettuare interconnessioni di circuiti che comunicano dati in analogico e in digitale.
Qual è la soluzione migliore che dobbiamo adottare in questo caso? A questo punto del nostro progetto dobbiamo ragionare su qualcosa di aggiuntivo ma altrettanto importante: come funziona il nostro sistema, effettua una lettura continua o saltuaria? Cosa legge esattamente? L'accensione è tramite un pulsante? Funziona rimanendo in modalità IDLE tutto il tempo fino a quando non arriva uno stimolo esterno come ad esempio la moneta che è caduta? Il modo più semplice per realizzare questo progetto è sicuramente utilizzare un microcontrollore, svegliato su interrupt e che sia in grado di effettuare la lettura della variabile di interesse per poter incrementare il valore letto. Supponiamo, quindi, che in questa configurazione Arduino sia sufficiente a realizzare il nostro scopo, e che avendo soltanto un unico ponte di lettura, possiamo pensare di utilizzare come microcontrollore Arduino Nano.
Di seguito nella tabella le specifiche tecniche funzionali del microcontrollore scelto.
Bene, adesso abbiamo capito cosa viene letto e, potenzialmente, abbiamo anche deciso ogni quanto, dal momento che sarà soltanto una questione di programmazione introdurre più o meno dei ritardi nella lettura periodica.
A questo punto c'è bisogno di capire come il nostro sistema deve comunicare all'utente i dati rilevati. Quel portaspiccioli è un sistema che noi abbiamo in casa, al lavoro, in un luogo chiuso, non lo portiamo in giro con noi, quindi non ha bisogno di essere un dispositivo mobile. Di solito vogliamo sapere prima di prelevare se possiamo farlo, esattamente come accade quando andiamo in banca a prelevare dal nostro conto. Un modo interessante e semplicemente realizzabile per poter avere questa funzione a disposizione, è tramite l'utilizzo

di un display LCD. Per tutti quelli che programmano con Arduino è meglio che il display LCD si interfacci tramite I2C; la scelta viene fondamentalmente fatta per via del numero di collegamenti che è necessario realizzare tra il display e la scheda.
Con I2C avremo bisogno di soli 4 collegamenti, dei quali 2 saranno relativi all'alimentazione e due alla trasmissione dei dati, in particolare, uno sarà relativo alla vera e propria trasmissione del dato mentre l'altro ad un segnale di clock, utile per la sincronizzazione.
E se il dispositivo fosse a batteria? Quanto dovrebbe durare in base alla capacità della batteria? Quale dovrebbe essere il valore della sua tensione?
Come potremmo dimensionare un sistema di energy harvesting?

## E' TUTTO QUI?

Abbiamo così selezionato i componenti sulla base della funzione specifica che vogliamo realizzare.
Non resta che mettere in pratica queste idee.
Proviamo a pensare ora come potrebbe essere espanso il nostro progetto, quali altre funzionalità si potrebbero integrare e implementare, che tipo di complicazioni avrebbe questo sui progetti che vogliamo realizzare o sulle modifiche da fare a progetti già realizzati.
Supponiamo che il nostro portaspiccioli smart sia a casa e che noi vogliamo poter sapere da remoto quanti soldi abbiamo in un determinato istante in quel nostro piccolo deposito di sicurezza. Non sarebbe carino se ci fosse la possibilità di interrogarlo ad esempio tramite un BOT Telegram?
Questa è un'altra delle tante possibilità applicabili. Arduino Nano, tuttavia, è una scheda che non prevede interfacciamento con il web, non supporta alcun tipo di protocollo di comunicazione e tantomeno integra alcun tipo di radio transceiver.
Questo significa che se volessimo utilizzare il progetto nella sua forma corrente, avremmo due possibilità di configurazione a nostra disposizione:

1. collegare in qualche modo ad Arduino Nano, un'altra scheda dotata della capacità di interconnessione, che sia possibile connettere ad una rete come il Wi-Fi, per trasferire tramite essa i dati;
2. sostituire Arduino Nano con una scheda che già nativamente integra queste capacità indispensabili per poter eseguire le funzioni delle quali abbiamo parlato. Ovviamente, questa seconda strada è certamente più macchinosa, benchè molto più corretta dal punto di vista progettuale e funzionale.

Un progettista sa che questo tipo di interventi implicano inevitabilmente degli impatti sul budget. Sorgono cosi' ulteriori domande. Quanto costano i componenti aggiuntivi? Quanto tempo richiede la sostituzione? Quante righe di codice? Quanto consumerà il progetto quando sarà finito? Sarà ancora conveniente dal punto di vista economico dopo aver applicato le modifiche?
Ecco, queste sono solo alcune delle domande che è indispensabile porsi per poter comprendere che la prototipazione rapida propone l'incredibile vantaggio di semplificare l'accesso a queste ed a ben altre domande progettuali.
Naturalmente queste ultime hanno poi l'effetto di stabilire il nuovo indirizzo del nostro progetto.

## CONCLUSIONI

Con queste considerazioni siamo arrivati alla conclusione del nostro appuntamento odierno.
Questo primo esempio concreto dovrebbe avervi dato l'idea del fatto che la prototipazione rapida abilita dei ragionamenti estremamente complessi anche su tematiche apparentemente semplici.
Il vero vantaggio della prototipazione rapida resa disponibile a tutti tramite schede di prototipazione come Arduino, è la possibilità di interrogarsi su dinamiche complesse, meccanismi e automatismi che vengono abilitati dall'inclusione oppure dall'esclusione dell'uno o dell'altro componente. La bellezza della progettazione diventa quindi una sfida alla portata di tutti. Ma questo era solo un esempio, anche se molto significativo, perché per la realizzazione dell'oggetto scelto sono state implementate sia conoscenze inerenti l'ambito IoT, che lo studio della prototipazione rapida, il cui utilizzo apporta anche notevoli vantaggi, come la riduzione del numero complessivo di attrezzature di prova che entrano effettivamente in produzione solo quando è stato creato il prototipo vero e proprio. La prossima volta ci occuperemo di un'altra idea progettuale che servirà a dotare il vostro smartphone di ulteriori funzionalità avanzatissime. Non resta che darvi appuntamento al prossimo articolo.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/iot-e-chi-iot-lo-fa-il-portaspiccioli-smart

Electronic Components

tme.eu

# CONDENSATORI POLIMERICI IBRIDI IN ALLUMINIO

Panasonic
INDUSTRY

## Condensatori a basso ESR altamente efficienti ed affidabili

Conformi alla norma AEC-Q200

Electronic Components

TME Italia S.r.l.
Via Zanica 19K, 24050 Grassobbio (BG)
tel. +39 035 03 93 111
fax +39 035 03 93 112
tme@tme-italia.it

facebook.com/TME.eu
youtube.com/TMElectroniComponent
linkedin.com/company/1350565
instagram.com/tme.eu
twitter.com/tme_eu

www.tme.eu

# IOT È CHI IOT LO FA: UNO SMART STAND PER IL NOSTRO SMARTPHONE

di **Pietro Boccadoro**

*Bentornati al nostro appuntamento con la prototipazione rapida. In questo articolo continueremo a basare lo sviluppo delle nostre idee di progetto sui principi fondamentali del rapid prototyping e dell'Open Source. Questa volta, però, il caso di studio che vorremmo realizzare riguarda il nostro smartphone. In particolare, un sistema di supporto per rendere più automatizzati alcuni meccanismi tipici della domotica. Scopriremo che il prototipo potrà essere utile non soltanto in casa ma anche al lavoro. Siete pronti?*

## INTRODUZIONE

Ormai è chiaro: **la prototipazione rapida è un paradigma abilitante per la sperimentazione**. Le sue possibilità sono davvero infinite e consentono realmente di cambiare il nostro modo di ragionare e progettare. Dopo aver visto in generale cosa significa fare l'IoT, nell'ambito di progetti più o meno complessi, oggi proviamo a guardare con occhi diversi un altro oggetto di uso comune, sicuramente sulla scrivania di molti, ovvero lo stand del cellulare. Non sono molti quelli di noi che utilizzano lo smartphone senza uno stand che lo tenga sollevato sulla scrivania, accanto allo schermo del computer. Chiunque faccia un lavoro di ufficio sa che questa configurazione consente di rispondere in maniera più efficiente alle chiamate, diminuire il consumo della batteria, ottimizzare le pause e propone anche diversi altri vantaggi. Il problema è che lo stand del cellulare è un elemento puramente passivo, talvolta un elemento d'arredo ma sicuramente non propone nessun livello di interazione con l'utente. Serve a svolgere semplicemente la sua funzione, ovvero mantenere il cellulare in una data posizione. Eppure, anche un oggetto così semplice può essere trasformato in qualcosa di estremamente più complesso e che rientri a far parte di un ecosistema di automazione che migliora le condizioni di lavoro e, in definitiva, la vita di chi lo possiede.

## POSSIBILI SCENARI

Proviamo ad analizzare questo oggetto per capire esattamente come può funzionare e per cosa può essere utilizzato oltre ciò che naturalmente vediamo. Il cellulare ha una proprietà: viaggia insieme con l'utente, si sposta con lui, in sostanza vive la stessa realtà del suo possessore. Molto spesso il telefono consente di svolgere azioni nel mondo come ad esempio il pagamento al supermercato oppure al cinema. Quando l'utente è al lavoro, occupa il suo posto in scrivania, mantiene quell'assetto per diverse ore durante la giornata, e non è detto che non riceva anche telefonate. Probabilmente, quando l'utente arriva al lavoro, seduto alla sua scrivania, ha anche una connessione Wi-Fi alla quale collegarsi. Ciò propone anche dei vantaggi in termini di efficienza energetica, dal momento che consente all'utente di **limitare il consumo energetico** sfruttando proprio la connessione Wi-Fi piuttosto che la rete dati. Inoltre, una volta che si è giunti alla propria postazione, non dovendosi muovere più per diverse ore durante la giornata, si potrebbe sfruttare il tempo per ricaricare il cellulare. Queste sono possibilità che lo smartphone già offre e che si possono già sfruttare. Quando si arriva a casa, probabilmente si possiede un altro stand, che svolge il ruolo di deposito del cellulare durante il periodo di permanenza in ambito domestico. Questo ci suggerisce che lo stand possa essere in qualche modo collegato all'ambiente in cui il cellulare si trova. Non sarebbe bello se questo potesse diventare parte dell'intelligenza dell' oggetto, in chiave 2.0? Dovremmo avere a disposizione qualche forma di tecnologia che sia in grado di capire quando il cellulare si trova in prossimità dello stand e ovviamente quando se ne allontana. Ma qualcosa di questo tipo già

*Figura 1. Breakout board di Adafruit basata su PN532*

esiste: si tratta dell'NFC, acronimo di Near Field Communications. Si potrebbe anche pensare di utilizzare il Bluetooth. Quindi esistono almeno due possibili tecnologie sulle quali potremmo basare lo sviluppo del nostro prototipo. Entrambe, infatti, si rivolgono alle **interconnessioni** ed alle **interazioni a corto raggio**, sebbene il Bluetooth riesca a coprire distanze maggiori dell'NFC. Quest'ultima tecnologia, infatti, è rivolta alle comunicazioni a cortissimo raggio, ovvero non oltre i 10 cm, nel caso migliore. Il Bluetooth, invece, è una tecnologia di comunicazione che si rivolge a comunicazioni a corto raggio ma sempre nell'ambito di distanze leggermente più grandi, tipicamente della decina di metri. Analoghe considerazioni andrebbero fatte per la mole di dati gestita, che nel caso del Bluetooth è decisamente superiore rispetto a quanto accade con l'NFC. Queste differenze non sono importanti solo quando si vogliono valutare le prestazioni di una rete di telecomunicazioni, ma anche quando si vuole comprendere se la tecnologia sia più o meno utile, se la sua applicabilità sia valida o meno. Il caso specifico che stiamo considerando prevede il cellulare che viene poggiato sullo stand e lasciato lì fino a quando non arriva il motivo per cui l'utente, proprietario del telefono, debba spostarlo. Il telefono sarà appoggiato, ovvero si assume che la distanza sia davvero irrisoria. Ecco per quale motivo possiamo pensare che le Near Field Communications rappresentino il candidato ottimo per tale applicazione. Adesso che abbiamo a disposizione una tecnologia, sappiamo che il nostro cellulare deve essere abilitato NFC. Supponiamo di voler progettare uno stand per il luogo di lavoro, la rilevazione dell'NFC, in questo caso, dovrà semplicemente abilitare la rete Wi-Fi, già salvata in precedenza (e quindi l'autenticazione sarà automatica), spegnere la connessione dati, magari abilitare il Bluetooth per la connessione del cellulare al computer in maniera tale da poter scambiare dati facilmente e diminuire la luminosità dello schermo. Queste sono azioni abbastanza semplici che possono essere svolte anche con l'utilizzo di un'app elementare. Peraltro, di app che gestiscono tag NFC ce ne sono tantissime, per cui non vale la pena passarle in rassegna tutte. Abbiamo, quindi, capito che il caso in cui vogliamo semplicemente che lo stand identifichi il posto, si può risolvere facilmente con un solo tag collegato allo stand ed un cellulare abilitato con un controllo tramite app. Da progettare, qui, non c'è molto altro. Vediamo di complicare un po' le cose. Supponiamo di voler non soltanto programmare uno stand da ufficio ma anche uno da automobile ed uno da casa. Naturalmente si potrebbe pensare di gestire tutto tramite app nello stesso modo, ma la cosa comincia a complicarsi un po' quando invece di spegnere ed accendere i dispositivi all'interno dello smartphone, vogliamo fare qualcosa di più complesso, come ad esempio tenere traccia dei nostri spostamenti per poter creare degli ambienti di lavoro che siano più confortevoli ed un ambiente domestico più smart. Supponiamo, quindi, di voler mappare gli spostamenti di una persona da casa a lavoro per poter decidere se accendere o spegnere il riscaldamento oppure realizzare funzioni più complesse come, ad esempio, aprire automaticamente il cancello di casa una volta che si arrivi da lavoro con la macchina. Per poter implementare tutte queste funzioni diverse abbiamo sicuramente bisogno di un sistema più complesso che sia in grado di leggere i tag NFC, ma sia anche in grado di connettersi ad Internet per poter elaborare le informazioni in tempo reale. Immaginiamo, allora, che lo scopo del progetto sia seguire un utente durante i suoi spostamenti. L'utente esce di casa, togliendo il cellulare dallo stand domestico, per muoversi verso l'ufficio. Una volta uscito di casa con l'automobile, sarà disponibile la sua posizione GPS, per cui la triangolazione della sua posizione restituirà un punto non più all'interno delle mura domestiche. A cosa potrebbe servire tutto ciò? Beh, le luci di casa potrebbero essere rimaste accese, e comunque non ha senso tenere il

router Wi-Fi attivo quando non ci siamo, il riscaldamento sicuramente può essere abbassato in maniera tale da garantire un minor spreco energetico. Le variabili all'interno dell'ambiente domestico sono davvero tantissime. Aspettiamoci, sul luogo di lavoro, che il funzionamento possa essere riproposto nella misura in cui prima abbiamo identificato alcune funzionalità di interesse.
È possibile che tutto questo venga gestito attraverso un servizio web ma anche localmente. Volendo creare un ecosistema, è assolutamente indispensabile che ci sia un apparecchio in grado di eseguire alcuni compiti, interfacciandosi anche con il web, per poter "storicizzare" gli spostamenti, verificare le preferenze, consentire delle personalizzazioni ed anche, perché no, proporre delle configurazioni intelligenti all'utente.

## PROCEDIAMO CON ORDINE

Cerchiamo, se esiste, una soluzione tecnologica pensata per il rapid prototyping, che consenta di contenere i costi e abiliti lo stand alla comunicazione NFC. Uno dei più usati tra i chip NFC-ready è l'NXP PN532. Di breakout board e reference design che lo utilizzano ne troviamo diverse a disposizione, come ad esempio la board distribuita da Adafruit.
Trovato il componente che ci consente di fare la lettura e la rilevazione del nostro smartphone, dobbiamo comprendere che tipo di specifiche tecniche ci propone, in particolare dobbiamo studiare l'effettiva distanza di rilevazione, il tempo che intercorre tra una rilevazione e la successiva, ammesso che ci sia un vincolo su questo, le interfacce di comunicazione e, ovviamente, l'alimentazione.
Molte di queste specifiche potrebbero essere comuni a più esponenti della stessa categoria, ovvero è possibile che più reader basati su questa tecnologia, siano in grado di funzionare con la stessa tensione di alimentazione.
Non possiamo, però, esserne sicuri quando cominciamo, per cui una buona ricerca di mercato parte dal presupposto che stiamo cercando di comprendere tutte le possibilità che il nostro sistema di rilevazione propone. Una volta che abbiamo capito quali sono i valori notevoli, possiamo passare all'elaborazione dei dati. Il nostro reader, molto probabilmente, si occupa soltanto di effettuare la lettura, ma il trattamento del segnale e le successive azioni che ne derivano non sono comprese all'interno della scheda che stiamo osservando.
È necessario comprendere, dunque, quale sarà il circuito di interfaccia e quale sarà la logica del suo funzionamento. Una delle domande alle quali dobbiamo sicuramente poter rispondere è sul tipo di funzionamento che vogliamo per il nostro circuito. Vogliamo che sia sempre acceso? Una volta in funzione, il comportamento dovrà essere sempre lo stesso?
Queste domande sembrano banali ma ci aiutano a comprendere se stiamo cercando di progettare un sistema che potrebbe basarsi su un microcontrollore oppure se serva un microprocessore.

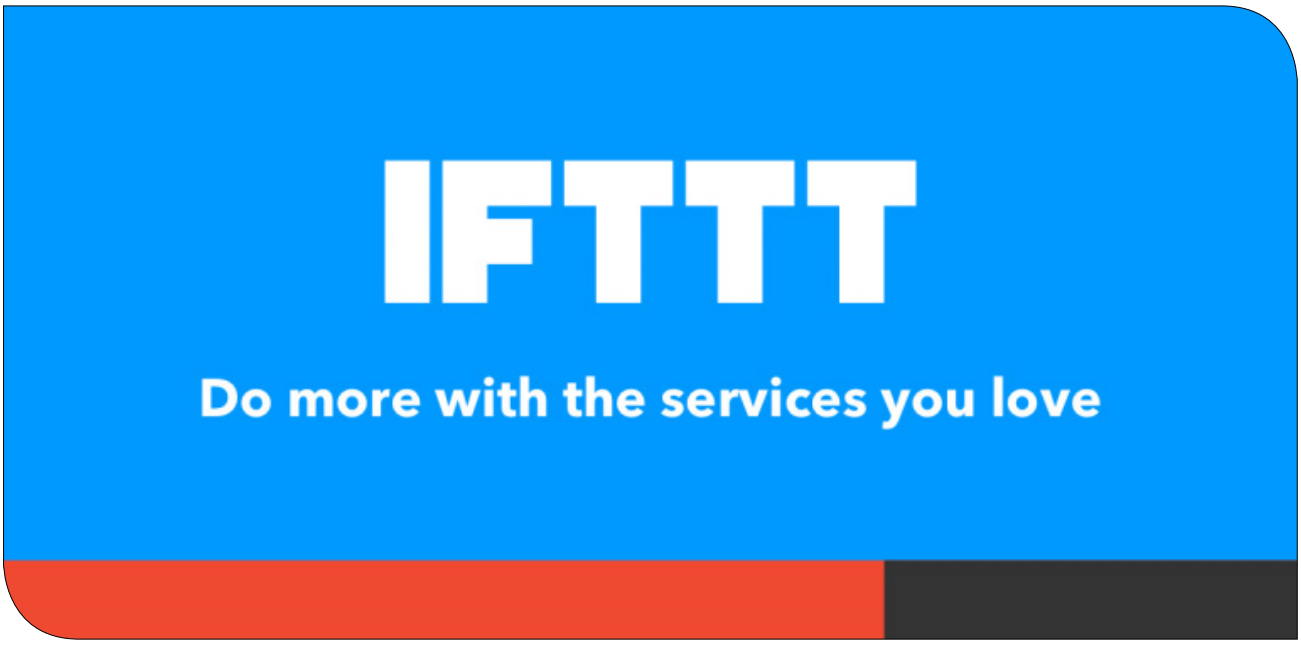



*Figura 2. If This Than That, IFTTT*

*Figura 3. Rappresentazione grafica delle interazioni abilitate da IFTTT*

**La differenza sta nell'intelligenza che dobbiamo mettere in campo, quella che ci serve per poter garantire che il nostro progetto svolga tutti i compiti assegnati nella migliore delle ipotesi e delle condizioni operative**. Se scegliamo un microcontrollore, possiamo cominciare a studiare la famiglia Arduino, sapendo che, oltre ai componenti ufficiali di questa scheda, esistono un esercito di cloni, una serie di varianti, e che comunque possiamo sempre prototipare la nostra soluzione. L'idea di partire con Arduino semplifica moltissimo i passi della prototipazione soprattutto per chi non ha grande esperienza.

Se, invece, noi abbiamo bisogno di gestire anche un database associato all'esecuzione del nostro software oppure preparare un web server locale per poter esporre dei servizi online, molto probabilmente abbiamo bisogno anche di un sistema operativo. In questo caso la scelta dovrà ricadere su una famiglia di schede che si basano su microprocessori, delle quali gli esponenti più di punta nell'ambito della prototipazione rapida sono certamente afferenti alla famiglia Raspberry Pi. Dal momento che il nostro sistema deve poter fornire anche consigli all'utente, è indispensabile superare il concetto del semplice microcontrollore e puntare direttamente con decisione ad una scheda che integri un microprocessore, che abbia una certa quantità di RAM ed una memoria all'interno della quale alloggiare tutti i dati misurati. Raspberry Pi risulta essere la scelta migliore. Proviamo, allora, a guardare il mercato e vediamo quali versioni esistono.

La prima, naturalmente, che ci viene in mente è la versione 3. Semplice, potente, compatta ed assolutamente funzionale. Possiamo utilizzarla per connettere componenti alimentati a 3.3 V e abbiamo a disposizione un header pin da 40 elementi che possono essere utilizzati per interfacciare circuiti digitali. Questa però è una cosa della quale dovremmo sicuramente tenere conto: Raspberry non consente l'interfacciamento analogico, ammesso che non si preveda introdurre anche un convertitore analogico-digitale aggiuntivo. Abbiamo scelto l'NFC, e ne siamo anche abbastanza convinti, soprattutto perché il Bluetooth non ci serve. Ma Raspberry Pi nativamente supporta anche le connessioni tramite Wi-Fi. Perché questo può essere utile? È presto detto: possiamo connetterci direttamente al web.

Raspberry consente di montare un web server al suo interno e di eseguire Apache e tutti i servizi che servono per la realizzazione di un sito internet. Quindi, con Raspberry Pi è possibile interfacciarsi direttamente al web, utilizzare servizi tipo IFTTT per poter automatizzare l'interconnessione ed i meccanismi di interazione

dell'utente con il sistema.
Cosa sarebbe cambiato se avessimo scelto Arduino? Nella sua versione Yun, ad esempio, Arduino propone già l'integrazione con il Wi-Fi. Sicuramente questo è un vantaggio perché non bisogna pensare di collegare una scheda esterna per garantire questo tipo di connessione.
E, tra l'altro, è anche un vantaggio dal punto di vista della memoria interna.
Ciò che Arduino Yun non ha, è sicuramente la capacità di elaborare i dati in maniera più intelligente, poter valutare le prestazioni del sistema, come viene fatto all'interno di un sistema operativo, soprattutto basato su Linux come Raspian.
Arduino Yun non potrebbe prevedere l'interconnessione di tanti componenti esterni quanti ne stiamo prevedendo in questo momento.
Inoltre, Arduino Yun per poter supportare l'interconnessione tramite Bluetooth, avrebbe bisogno comunque di un ulteriore componente. Insomma, il vero vantaggio dell'utilizzo di Raspberry è che si ha a disposizione una soluzione già molto completa, funzionale per tanti aspetti e sicuramente che richiede poca integrazione.
C'è anche però da valutare l'aspetto energetico. Una scheda Arduino è alimentata a 5 V, quindi consuma mediamente non meno di 500 mA, con alcuni picchi durante le trasmissioni di dati oppure quando è a pieno carico, anche 1.5 A, mentre Raspberry consuma di base 5 W.
Nel momento in cui si connettono tante periferiche via USB, si eseguono compiti complicati, si spinge il processore al suo limite, il consumo aumenta, la temperatura di esercizio anche, e si può incorrere in tutti i rischi connessi con le prestazioni che degradano come avviene in un computer normale.
**Dal punto di vista energetico, quindi, Arduino risulta essere una soluzione che garantisce un minor consumo energetico ed una certa ottimizzazione dello stesso**.
È anche vero, però, che Raspberry Pi può gestire tanti componenti diversi, e può arrivare a consumare anche 2 - 2.5 A.
Questo naturalmente non è di per sè un fatto negativo e va studiato **in funzione dell'applicazione specifica**.
A proposito di impatto energetico e footprint dei consumi, vale la pena specificare che **un progetto che rientra nella categoria dell'Internet delle Cose, ovvero dell'IoT, si prefigge come primo obiettivo, forse il principale, l'abbattimento totale dei consumi energetici**.
Si tratta di progetti che devono essere operativi consumando un valore di picco inferiore ai 50 mA, possibilmente alimentati con batterie LiPo da 3,7 V.
Qualora si trattasse di un alimentazione in logica TTL 5 V, staremmo comunque parlando di 250 mW di picco. Decisamente, un valore estremamente inferiore a quelli di cui abbiamo parlato fino a questo momento.

> ***"Cosa vuol dire questo? Che non è possibile fare IoT con Raspberry Pi e/o con Arduino?***

Certamente no! Esistono meccanismi avanzati di gestione delle risorse energetiche, interrupt selettivi, politiche di duty cycling, scheduling delle operazioni, e tanto altro.
Quello che vuol dire questa considerazione è, però, che manca ancora a questo sistema la capacità di gestire le operazioni eseguite in maniera intelligente per poter ottimizzare i consumi in maniera tale che il valore di picco si verifichi in un breve intervallo operativo.

## CONCLUSIONI

Con questo articolo siamo giunti alla fine dei nostri appuntamenti dedicati alla progettazione, al design ed alla realizzazione di progetti smart in ottica IoT basati sulla prototipazione rapida.
Il tema principale è stato il metodo, il ragionamento ma anche la comprensione degli strumenti, nonchè la definizione degli obiettivi. Abbiamo scoperto insieme che il ragionamento, lo studio e la definizione di cosa esattamente vogliamo dal nostro progetto, rappresentano la chiave di volta per giungere al risultato, ottimizzando al massimo le risorse a disposizione.
Ed in questo il portaspiccioli è stato un esempio molto funzionale, non trovate?
Come sempre, se ci sono domande, discutiamone insieme nei commenti. Se avete proposte per i vostri progetti o volete leggere altri articoli di questo tipo, non avete che da chiedere. In bocca al lupo a tutti voi, progettisti e makers!

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/iot-e-chi-iot-lo-fa-uno-smart-stand-per-il-nostro-smartphone

# APPLICAZIONI TECNOLOGICHE DI RELATIVITÀ RISTRETTA

di **Maila Agostini**

*$E=mc^2$ : la famosa legge che esprime la teoria della Relatività di Einstein. Ma cosa è davvero la Relatività? A cosa serve realmente? La Relatività Speciale o Ristretta, una delle due teorie sulla Relatività, sembra quanto di più lontano possiamo pensare dalle nostre vite. Tratta di particelle che si muovono alla velocità della luce! In realtà è stato necessario usare la Relatività per sviluppare alcune delle tecnologie che usiamo tutti i giorni; in questo articolo, dopo una breve introduzione sulla teoria, vedremo come è stata utilizzata per realizzare questi strumenti che ora sono di uso comune... o quasi!*

## UN PO' DI STORIA

La Relatività è stata formulata da Einstein, che nella figura 1 vediamo in una foto risalente ai suoi 25 anni, nel 1905; l'idea era quella di formulare una teoria che spiegasse il fatto che la luce viaggia nello spazio vuoto a velocità costante, come si evince anche dalla soluzione delle equazioni di Maxwell. Questo risultato contrastava con il principio di relatività galileiano; la velocità, fino a questo momento, non era mai stata vista come qualcosa di assoluto, ma come una grandezza relativa al moto di un altro sistema. Ma allora la velocità della luce si muove con velocità c rispetto a...? La risposta più quotata era quella che supponeva l'esistenza di un mezzo incorporeo e trasparente chiamato etere *luminifero*. Questa teoria era particolarmente gradita, in quanto riportava il problema ad una situazione conosciuta: le onde meccaniche avevano bisogno di un mezzo per propagarsi, quindi probabilmente ne avevano bisogno anche le onde elettromagnetiche. **Ma muovendosi in un mezzo, ogni corpo avrebbe dovuto produrre un vento con velocità opposta a quella del moto del corpo**. La luce avrebbe subito questo vento, quindi avrebbe avuto velocità diverse a seconda della direzione rispetto al suolo terrestre.

Nonostante gli esperimenti, di cui vediamo un esempio in figura 2, non si è mai trovata una differenza di velocità! Per questo Einstein suppose che l'etere non esistesse e cercò di realizzare una teoria che non ne tenesse conto.

## DUE TEORIE

Cominciamo sottolineando una differenza: ci sono due teorie della Relatività, Ristretta e Generale. La differenza tra le due è il sistema di riferimento. La Relatività Ristretta si occupa di moto di oggetti in sistemi di riferimento inerziali (che si muovono con velocità costante uno rispetto all'altro); in questo caso, le forze gravitazio-

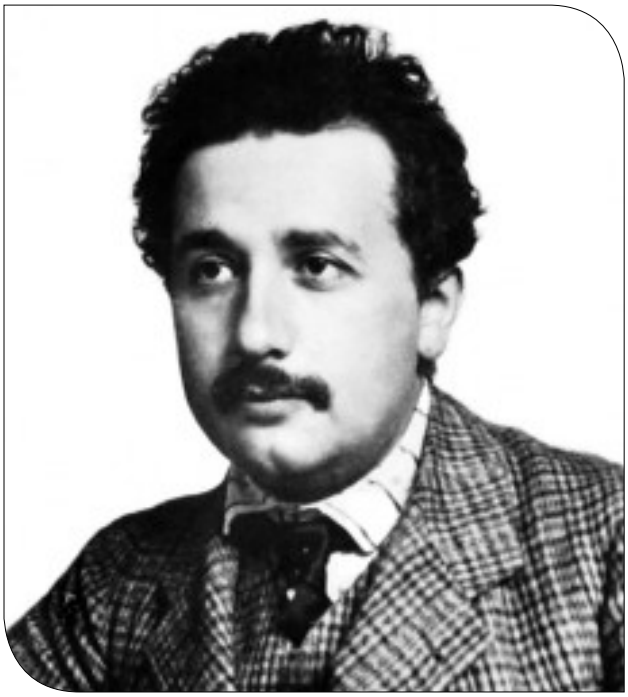


*Figura 1: Foto di Einstein nel 1904, scattata dal suo amico Lucien Chavan*

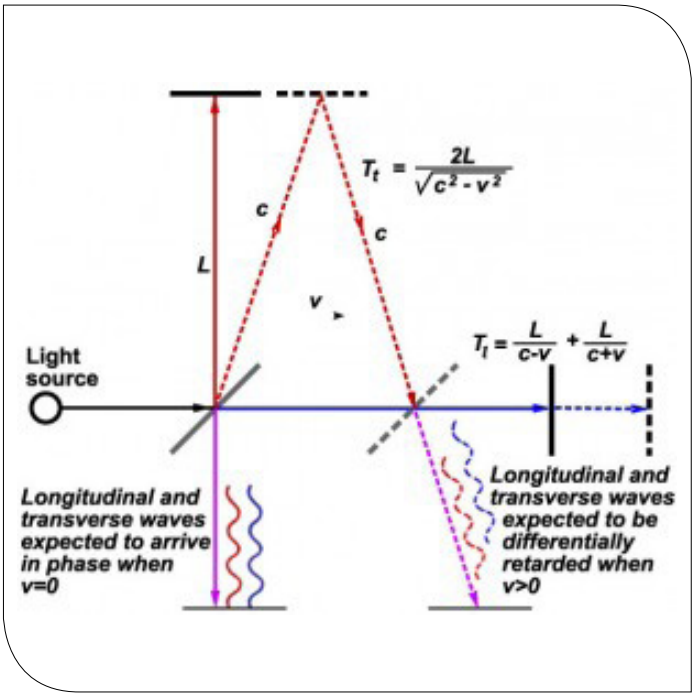



*Figura 2: Esperimento di Michelson e Morley*

nali sono nulle o deboli. La relatività Generale invece si applica per campi gravitazionali importanti, come quelli che creano le onde gravitazionali, recentemente scoperte (vedi link); in questa teoria, la gravità è vista come una proprietà dello spazio-tempo, curvato dalla presenza di massa o energia. In questo articolo ci occuperemo di applicazioni di Relatività Ristretta.

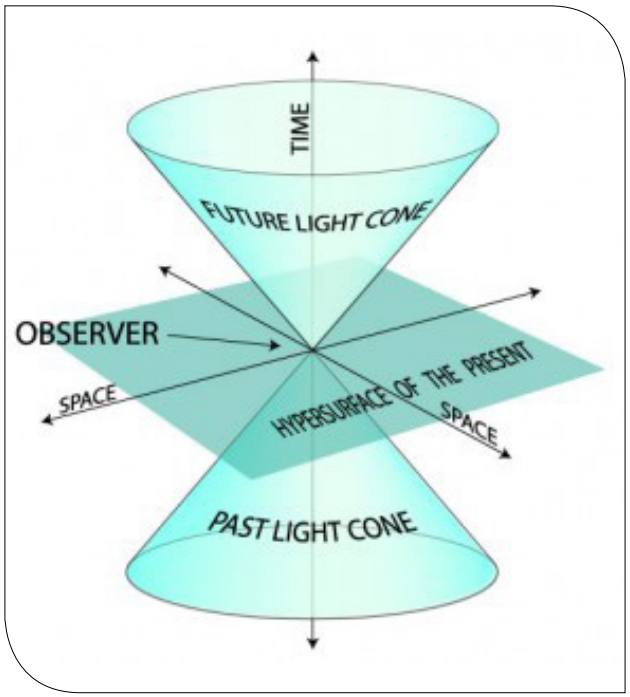



*Figura 3: Rappresentazione dello Spazio-Tempo relativistico*

## PRINCIPI FONDAMENTALI

Gli effetti della Relatività Ristretta si notano solo a velocità ed energie prossime a quelle della luce; a basse velocità la meccanica newtoniana, utilizzata fino ai primi del Novecento, è un'ottima approssimazione di questa teoria. Vediamo quali sono gli effetti di alte velocità od energie.

## SPAZIO-TEMPO

Lo spazio ed il tempo assoluti non esistono più; sono un'unica entità, di cui abbiamo una rappresentazione nella figura 3, e si influenzano tramite la velocità.

Per un corpo che si muove e velocità relativistiche possiamo osservare due fenomeni:

- una contrazione delle lunghezze **nella direzione di moto**, di un fattore $\gamma=1/(1-v^2/c^2)^{1/2}$ (v è la velocità dell'oggetto in movimento).
- una dilatazione temporale, sempre di fattore γ.

Inoltre, gli eventi simultanei per un osservatore possono non esserli per un altro osservatore.

## ENERGIA E MASSA

La massa di un corpo non è più data dalla proporzionalità tra forza ed accelerazione; la massa è solo l'energia a riposo di un corpo. Un oggetto in moto aumenta la sua massa di un fattore γ; quindi, se un corpo si muove alla velocità della luce, la massa tende a diventare infinita e così anche l'energia. Per questo motivo, un oggetto dotato di massa non può viaggiare alla velocità della luce, in Relatività Ristretta. La massa può essere convertita in energia; la conversione è caratterizzata dal *difetto di massa*. Se osserviamo la fusione di due nuclei, possiamo verificare che la massa totale è minore della somma della massa dei due; la differenza tra queste due quantità è appunto il *difetto di massa*, cioè la quantità di massa convertita in energia durante la trasformazione.

## APPLICAZIONI

Nonostante la Relatività sia un campo abbastanza giovane della fisica, esistono già numerose tecnologie per le quali è necessario tenere conto di correzioni dovute a questa teoria. Vediamone alcune.

## DISPOSITIVI A RAGGI CATODICI

Le apparecchiature a tubo catodico sono ormai già obsolete; si tratta della tecnologia che ha permesso il funzionamento dei vecchi televisori e schermi per pc.

Analizziamo il funzionamento (schema in figura 4): consideriamo un campo elettromagnetico. Vicino ad un fa-

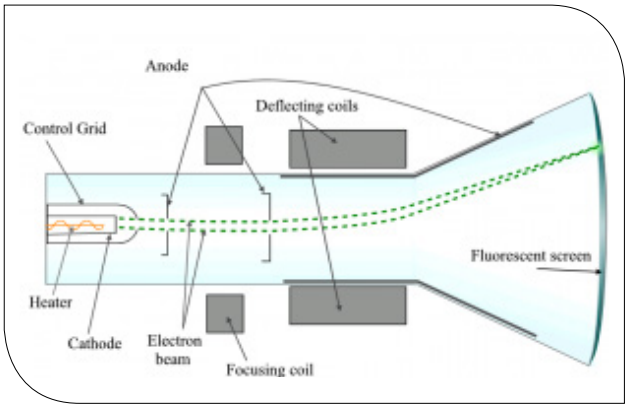



*Figura 4: Tubo a raggi catodici*

*Figura 5: Rappresentazione artistica di un satellite GPS*

scio di elettroni prodotto da un tubo a raggi catodici, posizioniamo parallelamente un filo metallico percorso da corrente, nel quale gli elettroni corrono ad una velocità molto inferiore a quelli nel tubo catodico. Nel sistema di riferimento del laboratorio, la deflessione a cui è sottoposto il raggio nel tubo catodico è dovuta al campo magnetico, quindi alla forza di Lorentz. Consideriamo invece un sistema di riferimento solidale con il treno di elettroni nel tubo catodico; in questo caso gli elettroni sono fermi, quindi non risentono della forza di Lorentz. Ma il fascio viene deflesso ugualmente! Nel filo sono presenti ioni positivi, in quiete rispetto al sistema del laboratorio, ed elettroni di conduzione, che rappresentano un flusso di cariche negative; rispetto al sistema di riferimento con il treno di elettroni nel tubo catodico, la velocità degli elettroni nel filo è più bassa degli ioni positivi. Per l'effetto di contrazione delle lunghezze, la distanza tra due ioni positivi consecutivi è minore rispetto a quella di due elettroni consecutivi. Il filo apparirà all'osservatore solidale col treno nel tubo più popolato di cariche positive che negative; quindi sembrerà carico positivamente. Quindi è il campo elettrico del filo il responsabile della deflessione del fascio. **In pratica, per un osservatore solidale con un sistema inerziale, la deflessione è data dal campo magnetico prodotto dal filo percorso da corrente, mentre per un osservatore relativistico, è data dalla contrazione delle lunghezze e dal campo elettrico del filo**.

### *GPS*

Ormai praticamente tutti i cellulari possiedono un localizzatore GPS (Global Positioning System). Questo sistema, rappresentato in figura 5, rileva la posizione in cui ci troviamo grazie al segnale di una costellazione di satelliti geostazionari, orbitanti a circa 20000 km da terra. Per ora è presente una costellazione di 31 satelliti, alcuni dei quali supplementari; questi satelliti ridondanti servono per aumentare la precisione della misura. Attualmente il sistema è non uniforme, una caratteristica che lo ha reso maggiormente affidabile, soprattutto in caso di guasti di più satelliti in contemporanea.

*Principio di funzionamento*

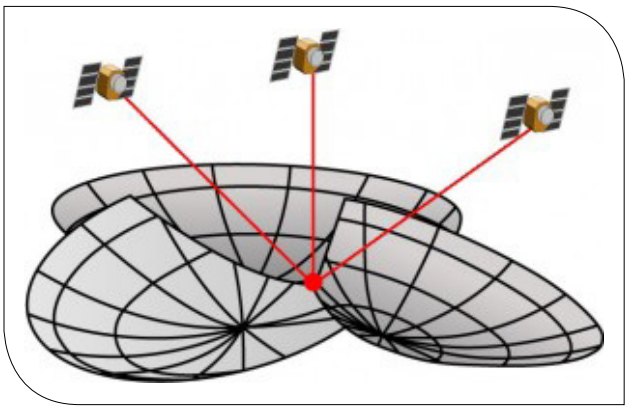


*Figura 6: Trilaterazione di un sistema GPS*

Il principio di funzionamento è relativamente semplice: si tratta di un sistema di trilaterazione sferica, come vediamo in figura 6, che parte dalla misura del tempo impiegato da un segnale radio a percorrere la distanza satellite-ricevitore. Il ricevitore calcola le sue coordinate a partire dalla posizione e distanza di almeno tre satelliti; questi trasmettono ogni 30 secondi le loro coordinate e il tempo di inizio della trasmissione. Il navigatore misura il tempo impiegato dal segnale per arrivare a terra tramite un orologio al quarzo, incorporato nello strumento, e lo moltiplica per la velocità della luce, trovando così la distanza tra ricevitore e satellite GPS. Si tratta di un sistema di trilaterazione sferica in quanto, a partire dalle distanze note dei satelliti e dalla loro disposizione,

è possibile tracciare tre sfere, ognuna centrata su un GPS, con un raggio pari alla distanza dal navigatore; poiché le sfere si intersecano in massimo due punti, di cui uno solo sulla terra, abbiamo finalmente le coordinate del punto dove si trova il navigatore.

*Misure di distanza, misure di tempo*

*Figura 7: Orologio atomico con apparati di supporto*

Sembrerebbe quindi che si possa ottenere una buona precisione da una misura di tempo, ricavata grazie ad un orologio al quarzo da pochi euro. In realtà non è così. Su ogni satellite si trova un orologio atomico, di cui abbiamo un esempio in figura 7, sincronizzato con tutti gli altri. Utilizzando i dati di un quarto satellite si può stimare la differenza tra l'ora segnata dai satelliti e quella segnata dal navigatore. Infine, misurando la distanza del quarto satellite tramite il calcolo della posizione rispetto agli altri satelliti, si ha la correzione per l'orologio del navigatore. Ma...dov'è la Relatività? I satelliti GPS si muovono circa alla velocità di 4 km/s; la Relatività Speciale afferma che a bordo del satellite si ha una dilatazione temporale pari a γ. In realtà si tratta di una differenza di **due decimi di miliardesimo**! Il problema è che l'effetto si accumula nel tempo, e dopo solo un giorno di volo, la differenza tra i due orologi è maggiore di **sei ordini di grandezza**! Questo ci porta ad un errore sulle distanze di ben **6 km**! Tuttavia, nel 1977, questo effetto era ancora poco conosciuto; quando il primo satellite GPS di prova fu messo in orbita, con una correzione errata ci fu più di un problema nel rilevamento delle coordinate del ricevitore!

## ACCELERATORI DI PARTICELLE: IL CICLOTRONE

Il ciclotrone è un acceleratore le cui particelle accelerano verso l'esterno dal centro di un percorso a spirale; è utilizzato soprattutto per la produzione di radionuclidi in medicina nucleare.

*Principio di funzionamento*

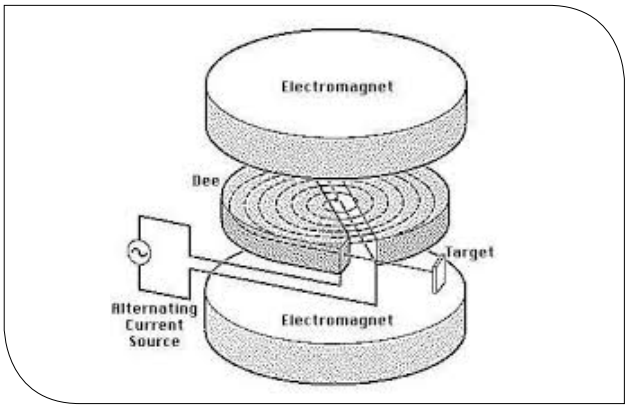



*Figura 8: Schema di funzionamento del ciclotrone*

In questo acceleratore, come vediamo in figura 8, abbiamo un campo magnetico uniforme, perpendicolare al piano della spirale, ed un campo elettrico oscillante tra due elettrodi cavi.
Una particella carica nel piano della spirale si muoverà di moto circolare uniforme. Nel campo magnetico, la particella è soggetta alla forza di Lorentz, che è perpendicolare sia al campo **B** che alla velocità; la forza agente è quindi centripeta, così come l'accelerazione.
Ma l'unico effetto di una accelerazione centripeta è quello di modificare la direzione della particella, non il modulo della sua velocità.
Quindi, anche la velocità angolare rimane costante.
A modificare la grandezza della velocità ci pensa il campo elettrico tra gli elettrodi; durante il passaggio tra gli elettrodi, la particella riceve un'accelerazione che fa aumentare la velocità ad ogni passaggio.
In questo modo le particelle percorrono una traiettoria a spirale.
**Ma se le particelle raggiungono velocità di un paio di ordini di grandezza sotto la velocità della luce il meccanismo non funziona più!**
La velocità angolare comincia infatti a diminuire al crescere della velocità!

*La Forza nella Relatività Speciale*

Sappiamo dalla Relatività Speciale che quando una velocità si avvicina a quella della luce, la legge di Newton ha bisogno di una correzione, data dal solito fattore γ. **Quindi, uguagliando la forza centripeta con quella di Lorentz otteniamo la nuova relazione per la velocità angolare**:

$\omega=(qB)/(m\gamma)$ dove ω è la velocità angolare

q è la carica della particella, m è la sua massa

B è il modulo del campo di induzione magnetica presente nel ciclotrone

γ è il fattore relativistico.

Per come è definito, γ cresce con la velocità, e tende ad infinito quando le velocità sono confrontabili con quella della luce. Il ciclotrone perde quindi il sincronismo e non riesce più ad accelerare le particelle.

## CONCLUSIONI

Inizialmente, dopo che l'esperimento di Michelson e Morley aveva negato l'esistenza dell'etere, si prospettarono tre strade diverse per la fisica:

- le leggi dell'elettromagnetismo valgono solo in un sistema di riferimento privilegiato;
- le equazioni di Maxwell necessitano di una modifica;
- le leggi della meccanica necessitano di una modifica.

**Einstein scelse la terza strada, giudicata la più "fantasiosa". La comunità scientifica ci mise un po' ad accettare le sue teorie, a causa della loro natura rivoluzionaria**. A volte le teorie scientifiche sembrano essere molto lontane dalla realtà... ma senza di esse non esisterebbe nessuna tecnologia oggi!
Non è detto che una scoperta o una teoria siano immediatamente applicabili; ci vuole del tempo per capire ed accettare "nuove" realtà.
Ma la scienza è al servizio del progresso e del benessere dell'uomo. Sta al singolo decidere quale uso fare della nostra conoscenza!

## BIBLIOGRAFIA:


- Dispense di Teoria della Relatività, Mario Calvetti ver.2012-2013
- Relativity, Albert Einstein (ebook, 2008)


L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/applicazioni-tecnologiche-di-relativita-ristretta

# CONTROLLO REMOTO DI ARDUINO DA DISPOSITIVI MOBILE

di **Salvatore Civiletti**

*In rete è disponibile un ricco campionario di materiale relativo alla comunicazione tra Arduino e dispositivi mobile; spesso però non tutte le informazioni sono corrette e dettagliate, e muoversi all'interno dei vari forum alla ricerca di soluzioni può risultare una pratica sfiancante. La soluzione proposta in questo articolo è una rivisitazione nota ai più che si basa sull'utilizzo del formato OSC.*

## INTRODUZIONE

Il progetto proposto rappresenta un valido strumento per approcciare al mondo della domotica. Attraverso un'applicazione che gira su un dispositivo mobile ci si propone di controllare dispositivi connessi ad Arduino da qualsiasi parte del mondo. L'iterazione con gli oggetti virtuali dell'interfaccia (pulsanti, fader, etc.) causa l'invio di messaggi OSC che verranno utilizzati per caratterizzare il comportamento degli oggetti stessi, così ad esempio agendo su un Fader sarà possibile dimmerare il relativo canale RGB, piuttosto che modificare la frequenza dell'oscillazione utilizzata per il dimmeraggio automatico agendo sull'apposito controllo rotativo. Requisiti richiesti sono che sia Arduino che il dispositivo mobile siano connessi ad internet.

## ELENCO DEL MATERIALE UTILIZZATO

- Arduino UNO
- Ethernet Shield W5100
- Arduino IDE Release 1.0.6
- TouchOSC Editor e TouchOSC App
- Sony Xperia E2115
- Router ADSL + cavo Ethernet
- Libreria OSCuino
- Driver + alimentatore 12V
- Strip LED RGB 12V

## SCHEMA DI PRINCIPIO

Come si può osservare dalla figura seguente (figura 1) ad ogni oggetto dell'interfaccia dell'applicazione si può associare un "codice" *OSC* (vedremo più avanti cosa questo significhi), ed un range di valori numerici. Il comportamento associato agli oggetti va descritto opportunamente all'interno del codice da caricare su Arduino. Nello schema è possibile individuare: una funzione che si occupa della ricezione dei dati che pervengono dagli oggetti, e specifiche funzioni associate ad essi.

## COS'È L'OPEN SOUND CONTROL?

Per comprendere il funzionamento di questo progetto semplice, ma che nasconde al suo interno grandi potenzialità, è utile rispondere a questa domanda cercando di capire di cosa si tratta e come funziona senza però appesantire troppo la trattazione.

Più brevemente detto *OSC*, **l'Open Sound Control, è un "content format"** o formato. Questo significa che può essere visto e confrontato con formati come *XML, WDDX o JSON*. Esso infatti non definisce le caratteristiche tipiche di un protocollo, quali processi semantici come pattern di comando-risposta, ma solo un formato di messaggi. Perdoni il lettore l'eventuale abuso di linguaggio che potrebbe venire utilizzato all'interno di questo documento qualora ci si riferisca con il termine protocollo OSC va tenuto presente che si tratta di un protocollo solo in senso lato.

Sviluppato nel 1997 all'interno dei laboratori del dipartimento di Musica dell'Università della California, Berkeley da *Adrian Freed and Matt Wright*, questo formato ha visto una larga diffusione negli ultimi anni in diversi campi come la robotica, sistemi di musica professiona-

le (X32 Digital Mixing Console, DiGiCo SD9, SSL Sigma, etc), video light console (GrandMA2, Whole Hog 4, Martin serie M, ChamSys MagicQ, etc.), software come Resolume Arena, Ableton Live, Cubase, Logic PRO, QLAB, etc.

***"OSC consente di far colloquiare vari dispositivi multimediali, sia fisici che virtuali, all'interno di un network.***

Spesso contrapposto al protocollo *MIDI, OSC* non rappresenta un suo semplice antagonista. Una prima evidente e fondamentale differenza tra i due è rappresentata dalla natura della licenza che è di tipo proprietario per quanto riguarda il protocollo MIDI mentre è aperta per *OSC*. I due protocolli nascono con presupposti diversi. Il protocollo *MIDI* risulta sicuramente la scelta più indicata in termini di efficienza qualora si debbano interfacciare due o più apparecchiature. Nel mercato consumer le apparecchiature protendono verso l'utilizzo del protocollo *MIDI* e per questo motivo è ancora largamente diffuso. La semplicità di utilizzo del formato *OSC* all'interno delle reti lo ha invece reso indispensabile in apparecchiature professionali, che devono interagire efficientemente con altri dispositivi connessi al network.

Facendo riferimento alla documentazione fornita dalla CNMAT vediamo di capire come funziona *OSC* e come sfruttarlo per i nostri progetti.
Non essendo *OSC* un protocollo **non si occupa dell'invio dei dati**, che sono trasmessi in pacchetti. È la rete stessa che si occupa del trasporto e della consegna del pacchetto.
Ogni richiesta di operazione in uscita *OSC* da parte di una applicazione produce un pacchetto *OSC*, similmente a quanto accade con il protocollo *UDP*, che produce un datagramma *IP* da inviare (figura 2).

***"Esistono diverse librerie che consentono di gestire facilmente il formato OSC in Arduino. La libreria utilizzata in questo progetto è la OSCuino sviluppata e mantenuta dai creatori dello stesso protocollo ai laboratori del CNMAT (Center for New Music and Audio Technologies at the University of California at Berkeley).***

***"Il codice d'esempio riportato all'interno di questo documento fa riferimento a questa libreria.***

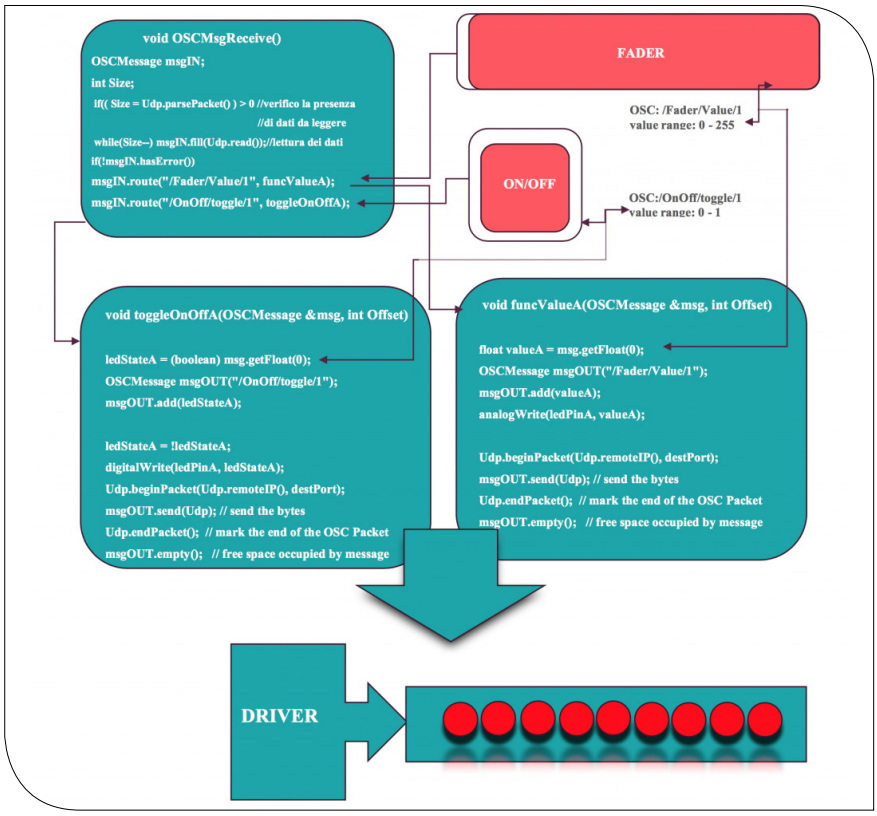



*Figura 1 : schema di principio*

Trattandosi di un formato che consente unicamente lo scambio di dati tra i vari client e il server per caratterizzare il comportamento di servizi personalizzati, si comprende facilmente il motivo per cui non esistano porte assegnate dallo *IANA (Internet Assigned Numbers Authority)*. Come si può vedere dalla figura 3 il formato OSC si colloca al livello 6 (presentazione) del modello OSI.

Un pacchetto *OSC* è costituito dal suo contenuto, un blocco contiguo di dati, e dalla dimensione su 8 bit del contenuto stesso. Il contenuto del pacchetto deve essere o un messaggio *OSC* o un *OSC Bundle* (questo viene specificato nel primo byte del pacchetto).

Un messaggio *OSC* (figura 4) è composto da un *Address Pattern* seguito da un *Type Tag* e da 0 o più argomenti (numeri e stringhe).

L'*address pattern* è l'equivalente dell'indirizzo IP, e segue la convenzione UNIX per la costruzione del percorso, quindi, deve sempre iniziare con "/". Il Type Tag invece definisce il tipo di dato. Nell'esempio *"f"* rappresenta un dato numerico a 32 bit in virgola mobile (float). Un *OSC Type Tag String* è una stringa che inizia con il carattere "," seguito da una sequenza di caratteri pari in numero al numero di argomenti del messaggio in questione e ne rappresentano una codifica del tipo di argomento (nella tabella 1 sono mostrati alcuni esempi). Ogni carattere dopo la virgola è chiamato *Type Tag* e rappresenta il tipo del corrispondente argomento. In figura 5 viene riportata la corrispondenza dei tipi comuni.

**OSC Bundle** è invece composto da una stringa: *"#bundle"* (8 byte), seguita da un *Time Tag* (8 byte), seguito a sua volta da 0 o più *OSC Bundle Element*. Un *OSC Bundle Element* è un intero (uint32 = 4 bytes) che indica la dimensione dei contenuti in byte, in multiplo di 4 byte (32 bit), seguito dai contenuti che possono essere sia un messaggio *OSC* che un *OSC Bundle* (i pacchetti possono contenere pacchetti).

I pacchetti ricevuti dal server sono passati a dei metodi. I metodi corrispondono a vari punti di controllo messi a disposizione. "Invocare" un metodo equivale a chiamare una "procedura", ossia passare argomenti al metodo causandone una risposta.

I metodi *OSC* sono organizzati in una struttura ad albero chiamata "*OSC Address Space*". I rami che si diramano sono detti "*contenitori*" e l'estremità del ramo farà capo ad un metodo. Ogni contenitore ha un nome simbolico, una stringa di caratteri *ASCII* che non deve contenere nessuno dei seguenti caratteri speciali (vedremo più avanti cosa significano):

**" ' # * , / ? [ ] { }**

Considerando ad esempio l'indirizzo OSC "/a/b/cde" secondo quanto esposto sopra avremo che:

- la *root* dell'albero presenta un contenitore: "a"
- il *contenitore* "a" contiene a sua volta un contenitore: "b"
- il *contenitore* "b" contiene il metodo o funzione: "cde"

Prendiamo adesso in considerazione il messaggio OSC con il seguente *address pattern*: "/oscillator/4/frequency" e supponiamo che presenti come unico argomento un float di valore 440.0. La rappresentazione del messaggio su 32 bit avrà la seguente forma:

```
2f (/)  6f (o)  73 (s)  63 (c)
69 (i)  6c (l)  6c (l)  61 (a)
74 (t)  6f (o)  72 (r)  2f (/)
34 (4)  2f (/)  66 (f)  72 (r)
65 (e)  71 (q)  75 (u)  65 (e)
6e (n)  63 (c)  79 (y)  0 ()
2c (,)  66 (f)  0 ()    0 ()
43 (C)  dc (Ü)  0 ()    0 ()
```

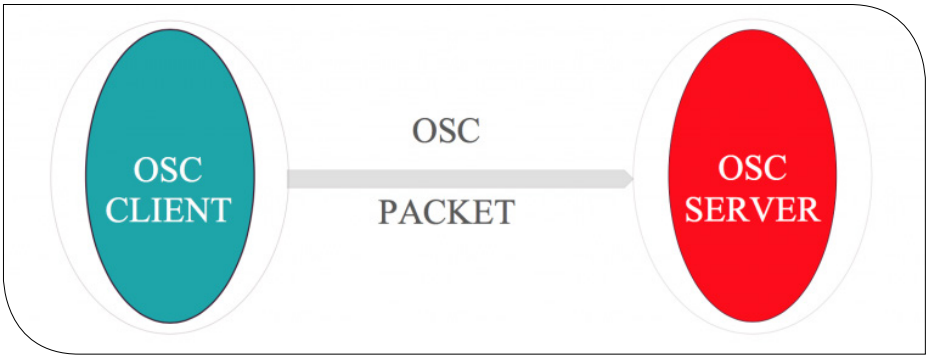



*Figura 2 : scambio di pacchetti OSC tra client e server*

*Figura 3 : posizionamento del formato OSC nel modello OSI*

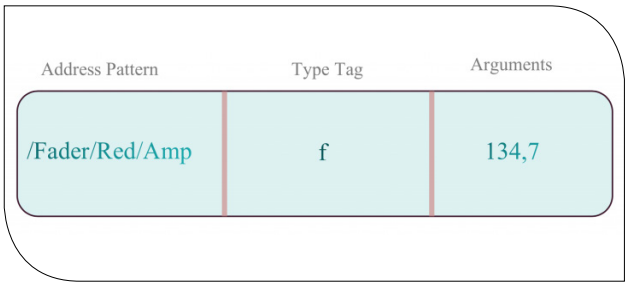



*Figura 4 : esempio di messaggio OSC*

Si noti che è stato riportando tra parentesi tonde la codifica *ASCII* del codice esadecimale che lo precede.
Se invece volessimo rappresentare il messaggio che ha come *adress pattern* "/foo" e come argomenti:

1. 1000 (int32)
2. -1 (int32)
3. "hello" (string)
4. 1.234 (float32)
5. 5.678 (float32)

Ciò che otterremmo in questo caso sono i seguenti 40 byte:

```
2f (/)  66 (f)  6f (o)  6f (o)
0 ()    0 ()    0 ()    0 ()
2c (,)  69 (i)  69 (i)  73 (s)
66 (f)  66 (f)  0 ()    0 ()
0 ()    0 ()    3 ()    e8 (è)
ff (ÿ)  ff (ÿ)  ff (ÿ)  ff (ÿ)
68 (h)  65 (e)  6c (l)  6c (l)
6f (o)  0 ()    0 ()    0 ()
3f (?)  9d ()   f3 (ó)  b6 (¶)
40 (@)  b5 (µ)  b2 (")  2d (-)
```

Alla ricezione di ogni messaggio, il server invoca i metodi appropriati al suo *OSC Address Space* (quando questo corrisponde con l'*OSC Address Pattern* del messaggio). Questo processo è chiamato ***matching***. La libreria *OSCuino* offre quattro metodi per gestire il pattern e l'*address matching*: *match(), fullMatch(), root()* e *dispatch()*.

***OSCMessage:***

*match* – verifica se c'è corrispondenza tra l'*address pattern* e l'indirizzo del messaggio. Restituisce il numero di caratteri che realizzano il matching.

```
OSCMessage msg("/a/0");
msg.match("/a"); // restituisce 2
```

*fullMatch* – restituisce *True* se dopo l'offset si ottiene un *matching* completo.

```
OSCMessage msg("/a/0");
msg.fullMatch("/0", 2); // Restituisce True
```

***OSCMessage && OSCBundle:***

*route* – invoca una funzione che ha come parametri l'istanza dell'oggetto *OSCMessage* inizializzata con la parte di indirizzo che realizza il matching con l'*address pattern* ed il numero di caratteri che combaciano.

```
 //definisco una funzione da richiamare quando si
ottiene il matching
 void routeCallback(OSCMessage & message, int
addressOffset){

    //esegue dei compiti...
    //con il messaggio riportato in basso l'addressOff-
set sarà uguale a 2.
 }
 OSCMessage msg("/a/0");
 ////////////////////////////////////////////////////////////////
 // 'route' usa 'match' per consentire matching parzia-
li                    //
 // invoca la funzione passando come parametri il
messaggio che esegue il matching  //
 // e l'address Offset
//
 ////////////////////////////////////////////////////////////////

 msg.route("/a", routeCallback);
```

| Tipo di argomento | OSC Type Tag String |
|---|---|
| Un tipo float sarà identificato da | ",f" |
| Due interi seguiti da una stringa, seguita da tre float corrispondono a | ",iisfff" |
| Se non ci sono argomenti avremo semplicemente | "," |
| Un argomento intero seguito a due argomenti di tipo blob corrispondono a | «,ibb" |

*Tabella 1 : Esempio di corrispondenza tra argomenti e OSC Type Tag String*

| OSC Type Tag | Type of corresponding OSC argument |
|---|---|
| u | uint32 |
| i | int32 |
| f | float32 |
| s | OSC-string |
| b | OSC-blob |
| T | True. * |
| F | False* |
| N | NULL* |
| I | Impulse* Unitype event trigger |
| t | OSC-timetag |

*No bytes are allocated in the argument data.

*Figura 5 : corrispondenza tra Type Tag e argomenti OSC*

*dispatch* – invoca una funzione se l'istanza dell'oggetto *OSCMessage* è inizializzata con un indirizzo che combacia pienamente con l'*address pattern*.

```
//definisco una funzione da richiamare quando si ottiene il matching
void routeCallback(OSCMessage & message )
{

  //esegue dei compiti...

  //con il messaggio riportato in basso l'addressOffset sarà uguale a 2.

}

OSCMessage msg("/a/[0-2]");
```

```
//////////////////////////////////////////////////////////////////////
// 'dispatch' usa 'fullMatch' quindi non sono possibili matching parziali.  //
// Invoca la funzione passando come parametro il solo messaggio che metcha.  //
//////////////////////////////////////////////////////////////////////

msg.dispatch("/a/1", routeCallback);
```

Come si è accennato esistono alcuni caratteri speciali che non possono essere utilizzati per la costruzione di un indirizzo ma servono per realizzare strutture di controllo del matching. Vediamo qual'è il loro significato e come utilizzarli:

1. **'?'** - Il matching è ottenuto da qualsiasi singolo carattere in quella posizione;
2. **'*'** - Il matching è ottenuto da qualsiasi sequenza di caratteri;
3. **'[ str ]'** - stringa di caratteri contenuta tra le parentesi quadre. Il matching è ottenuto da qualsiasi carattere della stringa. Inoltre se all'interno delle parentesi quadre:
   - due caratteri sono separati da un segno meno '-', ciò indica che la stringa è composta dal range di caratteri compreso tra i due (es. a-d equivale a "a,b,c,d");
   - il primo carattere è un punto esclamativo '**!**' , ciò indica una negazione della stringa che lo segue, in altre parole significa che il matching è ottenuto da qualsiasi carattere che non è contenuto nella stringa.
4. **'{ str1, str2 }'** – Lista tra parentesi graffe di stringhe separate da virgole. Il matching è ottenuto da qualsiasi stringa nella lista.
5. Per qualsiasi altro carattere che non rientra nelle

```
#include
#include
#include
#include
// MAC address della scheda Ethernet
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 0, 27 }; // Indirizzo IP della scheda Ethernet

int serverPort = 8083; // TouchOSC (incoming PORT)
int destPort = 8084;   // TouchOSC (outgoing PORT)

//Istanza EthernetUDP per l'invio/ ricezione di pacchetti UDP
int ledPinA = 3;
EthernetUDP Udp;

void setup(){
   Ethernet.begin(mac, ip); // Inizializzazione Ethernet

   ////////////////////////////////////////////////////////////
   // Inizializzazione del metodo Udp, sulla porta su cui verranno  //
   // trasmessi e ricevuti i pacchetti                              //
   ////////////////////////////////////////////////////////////

   Udp.begin(serverPort);
}

void loop(){
  ////////////////////////////////////////////////////////////
  // La comunicazione è costituita da un loop che dura finché ci sono  //
  // dati da ricevere e finché l'host è raggiungibile                  //
  ////////////////////////////////////////////////////////////

  OSCMsgReceive();
}

void OSCMsgReceive(){

 ////////////////////////////////////////////////////////////
 // Un OSCMessage può essere costruito anche senza un indirizzo ma  //
 // non risulterà valido finché non gliene viene assegnato uno.     //
 ////////////////////////////////////////////////////////////

 OSCMessage msgIN;
 int Size;

 ////////////////////////////////////////////////////////////
 // Prima di avviare la lettura dobbiamo assicurarci che i dati siano //
 // disponibili. Il metodo parsePacket() verifica la presenza di dati //
 // sulla porta di ricezione e restituisce la dimensione in byte del  //
 // pacchetto.                                                        //
 ////////////////////////////////////////////////////////////

 if(( Size = Udp.parsePacket() ) > 0 ){

 ////////////////////////////////////////////////
 // l'istanza msgIN viene riempita dal contenuto del //
 // pacchetto UDP fintantoché questo contiene dati   //
 ////////////////////////////////////////////////

 while(Size--) msgIN.fill( Udp.read() );

 ////////////////////////////////////////////////////////////
 // Se il messaggio non presenta errori e combacia con uno de     //
 // seguenti indirizzi OSC:                                       //
 // "/OnOff/toggle/1" or "/Fader/Value/1"                         //
 // vengono invocate le relative funzioni (eseguendole)           //
 ////////////////////////////////////////////////////////////

   if(!msgIN.hasError() ){  // se non ci sono errori

    msgIN.route("/Fader/Value/1", funValueA); //Fader CH Rosso
    msgIN.route("/OnOff/toggle/1", buttonStateA);//cambio label On Off
    msgIN. route ("/OnOff/toggle/1", toggleOnOffA); //Full/ Off CH Rosso

   }
 }
}
```

casistiche sopra esposte il matching è ottenuto solo dallo stesso carattere.

Nell'esempio che segue è stato utilizzato un *address pattern* strutturato utilizzando i caratteri speciali; si noti l'utilizzo dei metodi *match* e *fullMatch* ciò ci permette di comprendere come funzionano le regole sopra esposte.

```
void loop(){

    //setto l'indirizzo del messaggio nel costruttore

    OSCMessage msg0("/{input,output}/[0-2]/
[!ab]/*");

    //match restituisce il numero di caratteri che
combaciano

    int patternOffset = msg0.match("/input/1");  //
restituisce 8

    if( patternOffset > 0 ){
      if(msg0.fullMatch("/c/anything", patternOffset)){

    ////////////////////////////////////////////////////////////////////////////////
    // msg0.fullMatch("/c/anything", patternOffset)
verifica se il messaggio passato come   //
    // argomento a fullMatch realizza un match com-
pleto con l'indirizzo di msg0 a partire   //
    // dall'ottavo carattere (patternOffset) cioè verifica
se in qualche modo:            //
    //
//
    // "/[abc]/*" combacia con "/c/anything"
//
    //
//
    // secondo le regole dei caratteri speciali i due
pattern verificano un matching       //
    // verrà quindi eseguita la stampa sul monitor
seriale                          //
    ////////////////////////////////////////////////////////////////////////////////

      Serial.println("Match: /input/1/c/anything vs pat-
tern /{input,output}/[0-2]/[abc]/* ");

        }
    }
  }
```

Le specifiche del formato OSC indicano tre principali regole secondo cui vengono gestiti l'ordine e la priorità con cui i metodi devono essere invocati dal server.
Supponiamo che un server OSC includa metodi con i seguenti indirizzi:

- /first/this/one
- /second/1
- /second/2
- /third/a
- /third/b
- /third/c

1. Quando un pacchetto ricevuto contiene un singolo messaggio il server dovrebbe invocare il corrispondente metodo immediatamente.

2. Quando un OSC Address Pattern si traduce in un invio multiplo a metodi, l'ordine con cui vengono invocati i metodi non è specificato.

3. Quando un OSC Bundle contiene messaggi multipli il set di metodi deve essere invocato nello stesso ordine in cui si trovano i messaggi nel pacchetto.

Supponiamo inoltre che venga ricevuto un OSC Bundle che contenga i seguenti tre messaggi:

1. /first/this/one
2. /second/[1-2]
3. /third/*

Verranno invocati sei metodi nel seguente ordine:

- (1) : */first/this/one* dato che compare come primo nell OSC Bundle
- (2-3) : o */second/1* seguito da */second/2* o viceversa */second/2* seguito da */second/1*
- (4-6) : */third/a, /third/b, e /third/c* in qualsiasi ordine

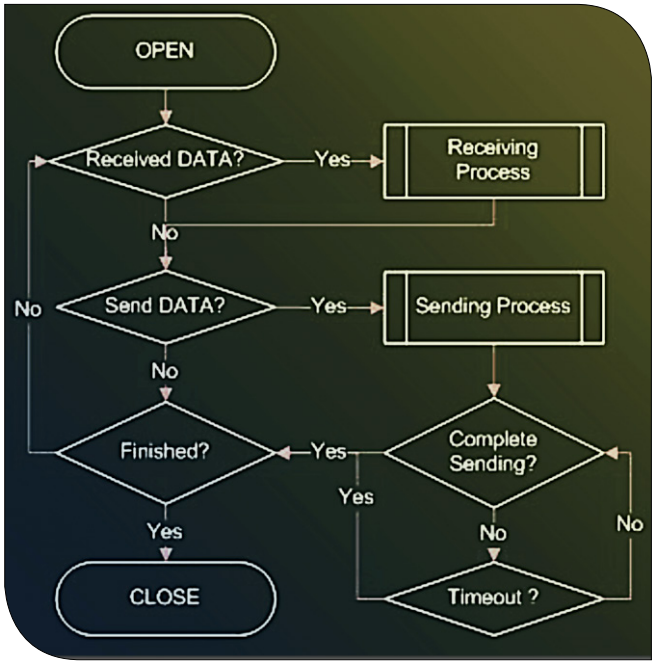



*Figura 6 : diagramma di flusso della comunicazione UDP*

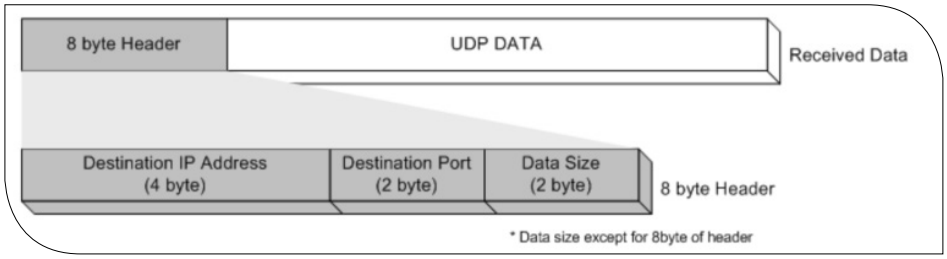



*Figura 7 : struttura del datagramma*

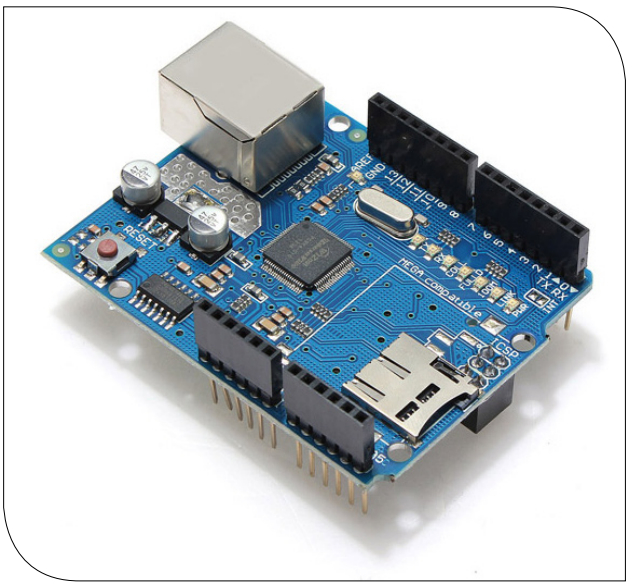


*Figura 8 : ethernet Shield w5100*

Avremo così dodici possibili modi in cui il server potrà invocare i sei metodi:

1. /first/this/one, /second/1, /second/2, /third/a, /third/b, /third/c
2. /first/this/one, /second/1, /second/2, /third/a, /third/c, /third/b
3. /first/this/one, /second/1, /second/2, /third/b, /third/a, /third/c
4. /first/this/one, /second/1, /second/2, /third/b, /third/c, /third/a
5. /first/this/one, /second/1, /second/2, /third/c, /third/a, /third/b
6. /first/this/one, /second/1, /second/2, /third/c, /third/b, /third/a
7. /first/this/one, /second/2, /second/1, /third/a, /third/b, /third/c
8. /first/this/one, /second/2, /second/1, /third/a, /third/c, /third/b
9. /first/this/one, /second/2, /second/1, /third/b, /third/a, /third/c
10. /first/this/one, /second/2, /second/1, /third/b, /third/c, /third/a
11. /first/this/one, /second/2, /second/1, /third/c, /third/a, /third/b
12. /first/this/one, /second/2, /second/1, /third/c, /third/b, /third/a

Quanto esposto finora ci consente di manipolare i messaggi *OSC*, ciò che manca per poter rendere tutto quanto funzionante è l'implementazione del meccanismo di invio e di ricezione dei messaggi. Attualmente il protocollo di trasporto maggiormente utilizzato con il formato *OSC* è di sicuro l'*UDP/IP*, ma va ricordato che **un pacchetto OSC può essere incapsulato in qualsiasi protocollo di comunicazione digitale**. Le specifiche caratteristiche di ogni protocollo di trasporto possono influire sulla qualità e la disponibilità dello stream dati a livello applicazione.
Dato che anche l'applicazione *TouchOSC* utilizza il protocollo *UDP* per lo scambio di messaggi *OSC*, diamo un breve sguardo al protocollo *UDP* per comprendere come gestire lo scambio di messaggi *OSC* tramite *UDP*.
L'*UDP* (*User Datagram Protocol*), è un protocollo a livello di trasporto ideale ad esempio per lo streaming video dato che garantisce comunicazioni a bassa latenza. Non viene però garantito né l'arrivo, né che l'ordine in cui vengono inviati i dati sia mantenuto in ricezione, inoltre non è previsto un meccanismo per il controllo degli errori.
Per la consegna dei pacchetti viene utilizzato il protocollo IP.
Di seguito sono elencate le fasi relative alla comunicazione *UDP*, mentre in *figura 6* è riportato il diagramma di flusso del che riassume tale processo che verrà implementato nello sketch.

1 – Creazione dei socket;
2 – Identificazione del socket;
3 – *Server* – si mette in attesa di un messaggio;
4 – *Client* – invio di un messaggio;
5 – Server – invio di risposta di avvenuta ricezione al *Client* (facoltativo);
6 – Chiusura del socket.

Il pacchetto *UDP* contiene il numero della porta sorgente, il numero della porta destinazione, la lunghezza totale del dato, il checksum che contiene il codice di controllo del datagramma: header + dati + pseudo-header quest'ultimo comprendente gli indirizzi *IP* di sorgente e destinazione (figura 7).

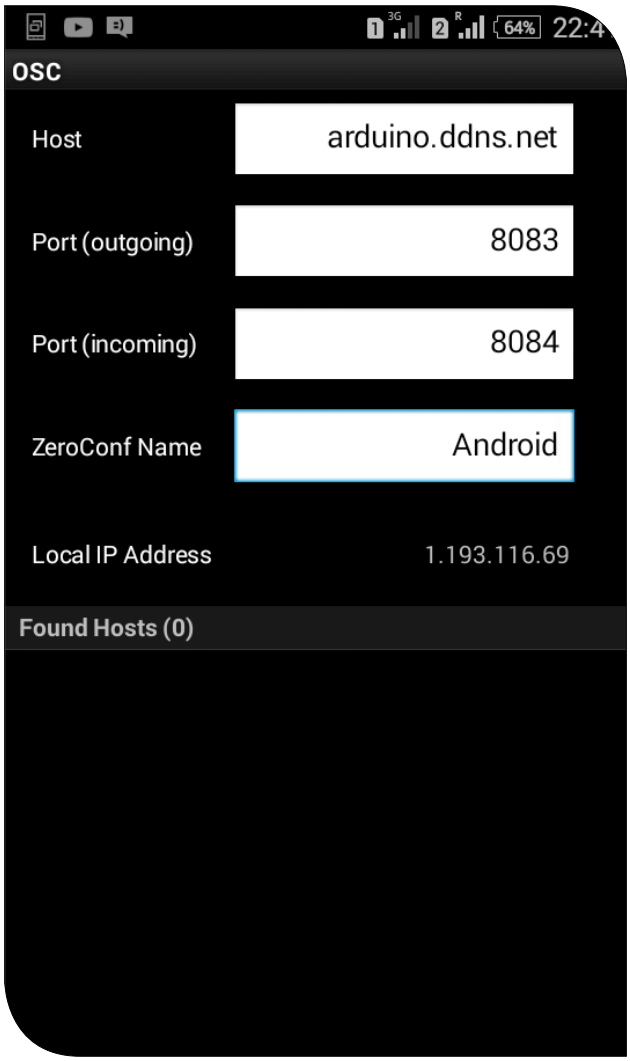



*Figura 9 : configurazione di TouchOSC*

Per realizzare la connessione di *Arduino* ad internet è stata utilizzata la *Ethernet Shield W5100* (*figura 8*), questa presenta anche la possibilità di utilizzare una scheda SD. La comunicazione tra Arduino, il chip Wiznet W5100 e la SD avviene attraverso il bus *SPI* (*Serial Peripheral Interface*) sui pin 10, 11, 12 e 13. Il pin 10 serve da *Slave Select* per il chip Wiznet, ovvero da segnale per selezionare la comunicazione con il chip *Wiznet*, mentre il pin 4 è utilizzato come *Slave Select* per la *SD card*. I pin 11, 12 e 13 sono utilizzati rispettivamente per *MISO*, *MOSI* e *CLK*. Risulta evidente che i pin 10, 11, 12 e 13 non potranno essere impegnati, e questo vale anche per il pin 4 qualora si dovesse utilizzare una scheda SD.

La comunicazione con il chip Ethernet e con la SD non può avvenire contemporaneamente e può essere ottenuta attraverso l'utilizzo della libreria *Ethernet.h* che mette a disposizione la classe *EthernetUDP* la quale consente di utilizzare il protocollo *UDP*. Come mostrato nell'esempio che segue è necessario creare un oggetto derivato dalla classe *EthernetUDP* che andrà inizializzato all'interno del blocco *setup()* con il numero della porta di ricezione del server. Nel blocco setup() occorre inoltre inizializzare la scheda *Ethernet* fornendo *MAC Address* e l'indirizzo *IP* della scheda *Ethernet*. Inoltre per verificare la presenza del pacchetto *UDP*, la classe *EthernetUDP* ci mette a disposizione il metodo *parsePacket()* il quale verificata la presenza del pacchetto e ne restituisce la dimensione (va chiamata prima di effettuare la lettura del buffer con il metodo *read()* ).

Nell'esempio sopra riportato si è visto come inizializzare la *Ethernet*, e come gestire la ricezione dei pacchetti *UDP* che contengono i messaggi OSC necessari per invocare la funzione associata. Questo significa che, agendo ad esempio sul fader (dall'interfaccia caricata sull'applicazione) che ha come indirizzo "*/Fader/Value/1*", verrà invocata la funzione funValueA ogni volta che con il dito scorriamo il cursore fino al rilascio (dato che *OSCMsgReceive()* è invocata un numero infinito di volte ma il messaggio viene inviato solo quando interagiamo con il fader producendo una variazione della sua posizione). Nella *funValueA()* verrà definito il comportamento del fader che in questo caso verrà usato per dimmerare il canale.

```
void funValueA(OSCMessage &message, int addrOf-
fset){

   //restituisce il dato memorizzato nella prima posi-
zione di message
   float valueA = message.getFloat(0);
   OSCMessage msgOUT("/Fader/Value/1");

   //aggiungo valueA al messaggio
   msgOUT.add(valueA);

   analogWrite(ledPinA, valueA);

   ////////////////////////////////////////////////////////////
   // Udp.beginPacket( remoteIP, remotePort);
//
   // avvia la connessione per scrivere il pacchetto
UDP sulla      //
   // connessione remota (TouchOSC)
//
   // remoteIP: indirizzo IP della connessione remota
(4 byte)      //
   // remotePort: porta della connessione (int)
//
   // restituisce 1 se avviene con successo 0 se ci
 sono problemi... //
```

*Figura 10 : esempio di interfaccia TouchOSC*

```
  // Udp.remoteIP() prende l'IP della connessione
remota (touchOSC) //
  ////////////////////////////////////////////////////////////

  Udp.beginPacket(Udp.remoteIP(), destPort);

  msgOUT.send(Udp); // invio del messaggio OSC
all'oggetto TouchOSC

  //endPacket() chiamato dopo la scrittura del dato
UDP
  Udp.endPacket();
  msgOUT.empty();   // svuota il messaggio

  }
 }
}
```

## CONFIGURAZIONI FINALI

Arrivati a questo punto è facile comprendere come funziona lo sketch e come è organizzato. Per la scrittura e la strutturazione dei propri programmi da caricare su Arduino e per la progettazione dell'interfaccia si suggerisce di procedere passo, passo partendo ad esempio prima da uno schizzo su carta annotando indirizzi e valori dei vari oggetti (fader e controlli rotativi, ad esempio, posso avere un range da 0 a 255 per realizzare un pieno controllo su 8 bit, mentre un range da 0 a 1 può essere associato ad un pulsante per gestire gli stati di ON e OFF). In seguito si può procedere, aggiungendo il primo oggetto, ad esempio un pulsante per la gestione degli stati ON e OFF, su TouchOSC Editor, e caricare l'interfaccia sul dispositivo mobile. Occorre quindi scrivere le funzioni per la gestione della ricezione del messaggio prima, utilizzando il meccanismo di matching, e poi la funzione di gestione (fortunatamente il LED si rivela un meccanismo di verifica immediato se collegato correttamente). Se qualcosa non funziona la seriale consente come sempre di effettuare le verifiche di debug. Se tutto funziona a dovere si può iterare questo processo passando all'oggetto successivo. Detto questo non resta che:

a- progettare l'interfaccia personalizzata con TouchOSC Editor.
L'interfaccia intuitiva rende quest'operazione semplice. Cliccando con il tasto destro all'interno dell'area di lavoro sarà possibile accedere ai vari oggetti. Come accennato ogni oggetto ha dei parametri che possono essere impostati, come l'indirizzo *OSC* e d il range di valori ad esso associato. Una volta che l'interfaccia è pronta, cliccando sul pulsante di sincronizzazione, sarà possibile eseguire l'upload all'interno dell'applicazione TouchOSC (dopo che vi si è impostato l'indirizzo *IP* del computer dal quale effettuare l'upload dal menu

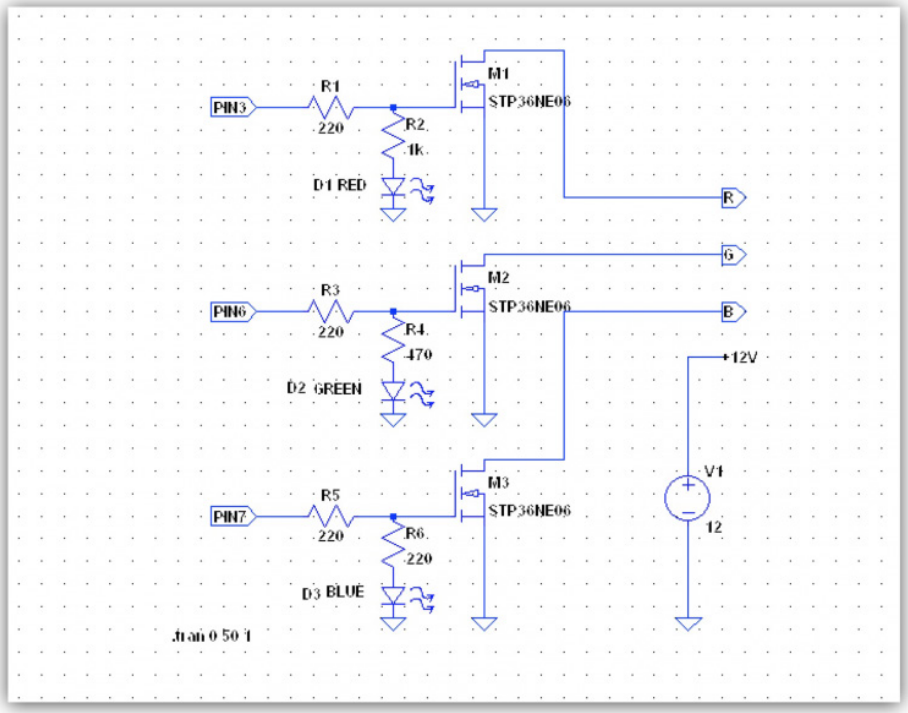



*Figura 11 : schema elettrico del driver*

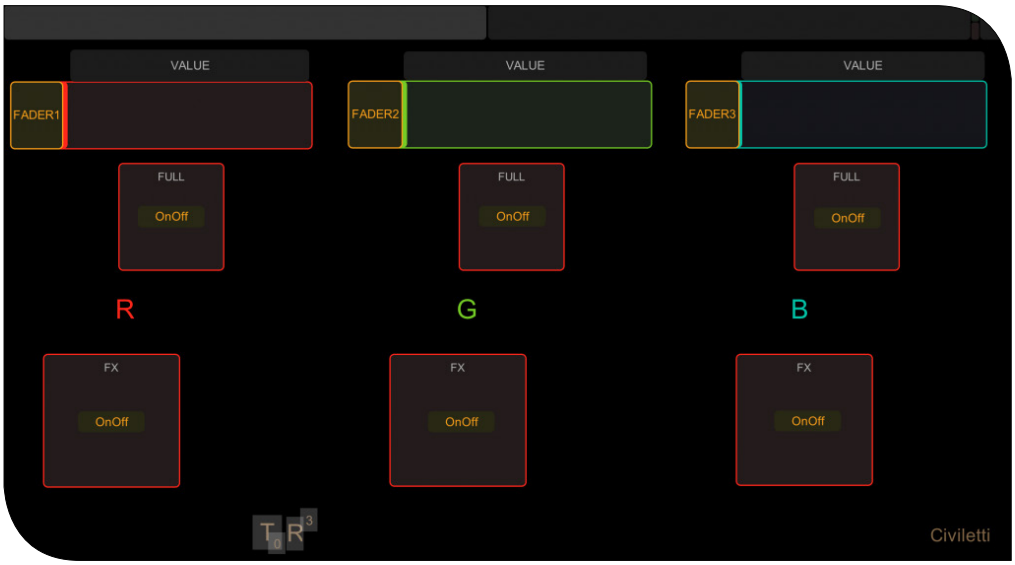



*Figura 12a: interfaccia ti test TouchOSC allegata al progetto - pagina 1*

layout). L'applicazione TouchOSC va inoltre configurata, come mostrato nella figura 9, assegnando l'indirizzo *IP* per raggiungere Arduino (vedi punto b), ed i numeri di porta (vedi punto c).Come mostrato nella figura 10 è possibile creare interfacce di una certa complessità.

TouchOSC Editor è scaricabile gratuitamente dal sito http://hexler.net mentre l'applicazione TouchOSC va acquistata.

b- Assegnare un *IP* statico utilizzando un servizio come NO-IP.
Questo consente di raggiungere il nostro dispositivo Arduino (che è connesso ad internet) dall'esterno della *LAN* senza correre il rischio che una qualsiasi anomalia che modifichi l'indirizzo esterno del router lo renda irraggiungibile.

c- Eseguire il port forwarding
Come sarà noto ai più questa operazione consi-

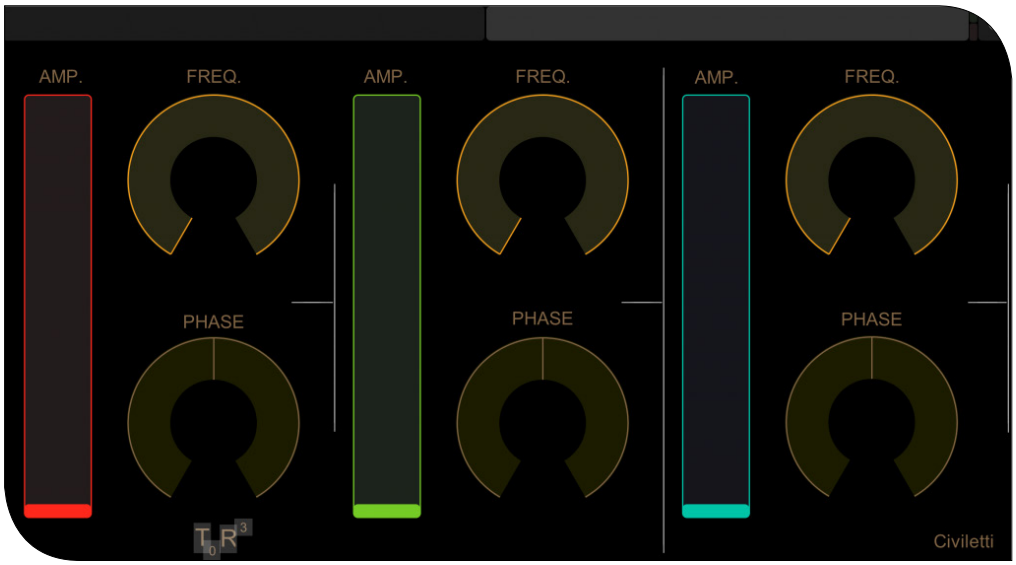



*Figura 12b: interfaccia ti test TouchOSC allegata al progetto - pagina 2*

ste nel "aprire un passaggio per i dati all'interno del firewall" per una determinata applicazione. Nel gergo informatico si dice "aprire una porta". Occorre accedere al proprio router per poter aprire le porte su cui vogliamo che avvenga la comunicazione *OSC*.

***"Nelle reti informatiche la PORTA è un percorso univoco per i dati che può essere utilizzato da un'applicazione alla volta.***

***"PORTA + IP = SOCKET***

***"L'indirizzo IP identifica il dispositivo in rete, mentre la PORTA identifica un'applicazione o più esattamente un protocollo***

In base al modello del router adsl in possesso una ricerca mirata potrebbe far scoprire diverse soluzioni per ottimizzarne il funzionamento.

d- Assegnare un *IP* statico alla *Ethernet Shield*

L'analisi condotta fino adesso fornisce tutti gli strumenti necessari per poter pilotare piccoli carichi, come dei LED o piccoli servo motori, a distanza.

Per completare il progetto occorre poter fornire al carico (strip LED) una maggiore corrente, in definitiva occorre un driver.

## SCHEMA DEL DRIVER

La realizzazione del seguente schema mostrato in figura 11 non richiede particolari abilità tecniche e con una spesa di pochi euro è possibile procurarsi tutti i componenti. Il circuito è stato testato con le strip LED RGB di uso comune, quelle cioè che richiedono un'alimentazione di 12V (più economiche di quelle che hanno una tensione operativa di 24V, normalmente destinate ad un utilizzo professionale), ma fornendo l'alimentazione opportuna è adatto a pilotare anche carichi maggiori.

*Approfondimenti:*

1. *Dispositivo di controllo per Strip Led RGB*
2. *Soluzioni di Driver per illuminazione a LED*
3. *Pilotaggio di un MOSFET di potenza con con un microcontrollore*

## CONCLUSIONI

Come si è visto nonostante la semplicità realizzativa di questo progetto sono stati messi in luce diversi aspetti interessanti.
Esistono applicazioni, come ad esempio *Blynk*, con cui la realizzazione di un sistema di controllo remoto simile a quello presentato è molto più semplice ed immediata, o come *IoT Manager* che offrono interfacce belle e pronte e lavorano con il protocollo *MQTT*.
Si suggerisce al lettore di provarle per trarne le proprie

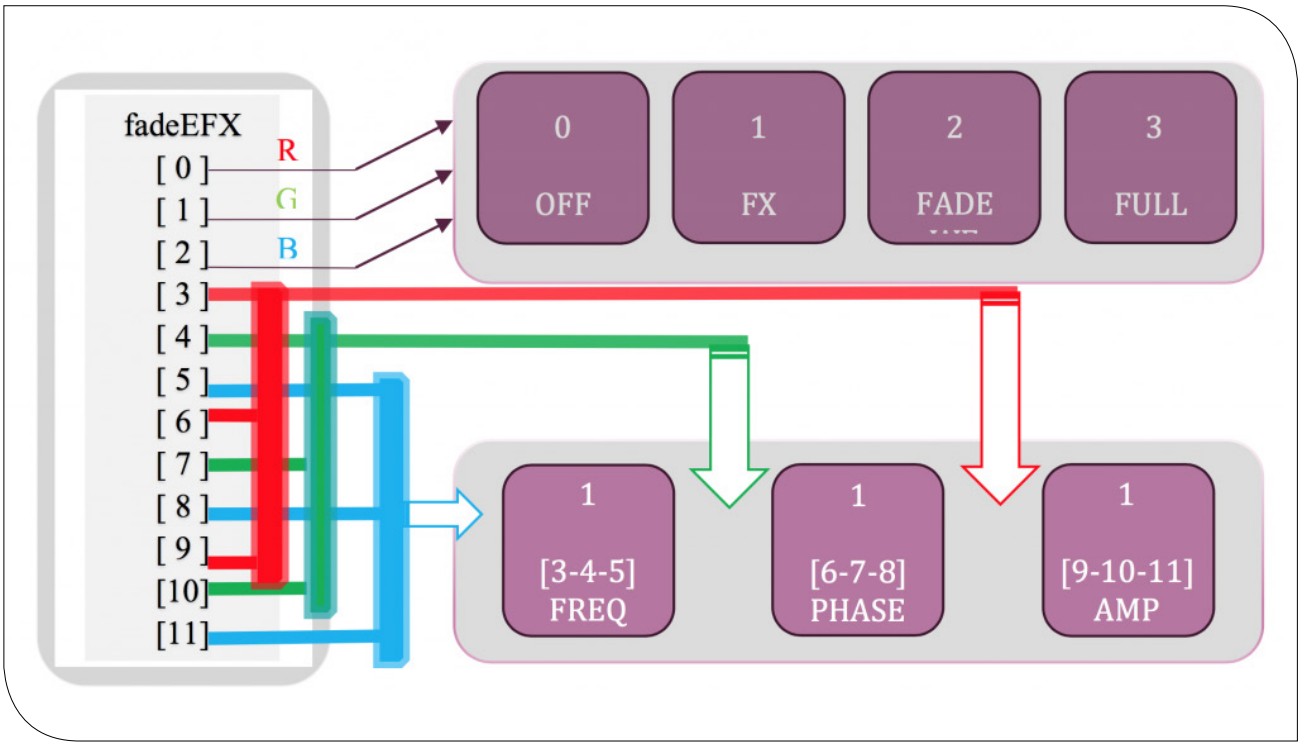



*Figura 13: controllo degli oggetti all'interno dello sketch allegato al progetto*

considerazioni. Il progetto è perfettamente funzionante ma può essere ovviamente perfezionato, inoltre estrapolando le componenti di maggior interesse lo si potrà personalizzare adattandolo alle proprie esigenze creative.

L'interfaccia TouchOSC prevede la creazione di layout su più pagine, si potrebbe sfruttare questa possibilità per creare ad esempio in ogni pagina uno schema per il controllo delle diverse stanze di un appartamento (accensione e dimmeraggio luci, accensione scaldabagno e riscaldamento, apri cancello etc.).

Ogni ampliamento di questo progetto farà sorgere diversi interrogativi la risposta ai quali potrebbe non essere immediata e banale, si spera di poterne mettere in luce alcuni in futuro fornendo possibili soluzioni.

In fine va precisato che per quanto riguarda l'applicazione, che fornisce oggetti in grado di gestire messaggi *OSC*, la ricerca inizialmente si è orientata verso applicazioni open source ma solo *AndrOSC* ha portato al raggiungimento di risultati apprezzabili.

Provi il lettore ad effettuare le proprie verifiche. Invito chiunque individui una qualche applicazione open source simile che funzioni in maniera appropriata di segnalarla; rivolgo inoltre un appello a chiunque sia in grado di sviluppare applicazioni Android perché si riapra il progetto *AndrOSC*.

## DOCUMENTI ALLEGATI

Dal link che segue è possibile scaricare il materiale, necessario per eseguire i test, ed i datasheet. In particolare viene fornito uno sketch in cui la porta seriale è abilitata per sfruttarla all'interno di alcune funzioni in modo da effettuare le verifiche che permettono di comprendere meglio quanto esposto.

Le figura 12a e 12b mostrano le due pagine dell'interfaccia TouchOSC allegata al progetto, ognuno degli oggetti richiamerà un'apposita funzione gestita attraverso il meccanismo schematizzato in figura 13.

Una volta compreso il funzionamento del sistema, per alleggerire lo sketch, si suggerisce di commentare le parti adibite al debug del programma.

*Documenti allegati*

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/controllo-remoto-di-arduino-da-dispositivi-mobile

# SEMPLIFICARE PROGETTI IOT BASATI SU STANDARD CELLULARI A BASSA POTENZA

di **Andrea Garrapa**

*La progettazione IoT basata sulla tecnologia LTE cellulare è complessa e ciò significa che ci sono molti percorsi da poter intraprendere per completare un progetto. Alcuni di questi percorsi potrebbero semplicemente aggiungere tempo e costi, come tutte le deviazioni non necessarie. O peggio, altri potrebbero significare non arrivare mai a destinazione con successo. L'obiettivo di questo articolo è quello di fornire dei suggerimenti in grado di aiutare gli sviluppatori a tracciare un percorso più diretto per implementare la connettività cellulare LTE a bassa potenza nei loro progetti.*

## INTRODUZIONE

Gli sviluppatori di prodotti che non vivono e respirano il design cellulare potrebbero essere incaricati di progetti di design IoT che richiedono una comprensione di come utilizzare i nuovi protocolli cellulari a bassa potenza **LTE CAT M1 e NB-IoT** in combinazione con tecnologie più familiari come il Bluetooth a basso consumo (**BLE**). Progettare funzionalità a basso consumo e connesse in un prodotto è difficile con qualsiasi tecnologia wireless, ma è particolarmente impegnativo con la tecnologia LTE cellulare (curva di apprendimento ancora più ripida), quindi è importante evitare di renderlo ancora più difficile prendendo il cammino più lungo.

In passato, la connettività cellulare per dispositivi alimentati a batteria come quelli utilizzati nelle distribuzioni IoT era poco pratica nella maggior parte delle situazioni; l'architettura cellulare e i protocolli comportavano costi

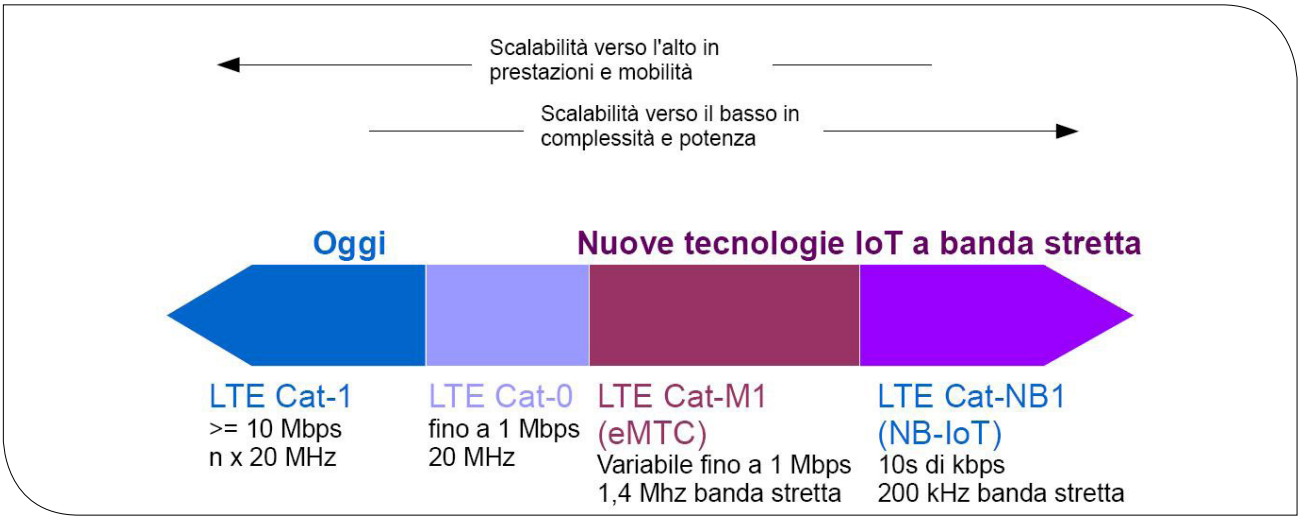



*Figura 1: scalabilità della tecnologia LTE con evidenziate larghezza di banda e bit-rate per M1 e NB-IoT.*

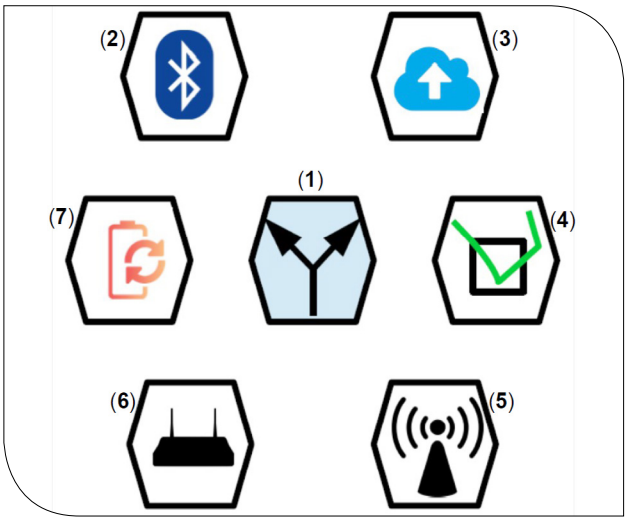



*Figura 2: i 7 aspetti da considerare nell'implementazione di un progetto IoT con standard cellulare (1)CAT-M o NB-IoT (2)Altri standard (3)Cloud (4)Test (5)Antenne (6)Gestore (7)Modalità sospensione.*

hardware troppo elevati e un'architettura radio "sempre attiva" che scaricava rapidamente le batterie. Quelle qualità "negative" potevano andare bene per dispositivi come gli smartphone, ma il costo elevato e l'elevato consumo di energia non erano adatti a progetti IoT come sensori wireless in impianti industriali, sensori ambientali e reti intelligenti. Tali svantaggi avevano messo in ombra i punti di forza della rete cellulare per l'IoT, tra cui l'ampia infrastruttura esistente, una specifica chiaramente definita e una tabella di marcia a lungo termine.

Ma due nuove versioni a basso consumo della tecnologia LTE — **CAT M1** e **NB-IoT** — sono state progettate per affrontare quelle sfide di inefficienza dei costi e dell'energia in modo da consentire di sfruttare i punti di forza dell'infrastruttura cellulare. In sostanza, c'è un bivio nel cammino di sviluppo LTE. Un percorso (verso il 5G) è ottimizzato per velocità molto elevate e latenze molto basse per applicazioni futuristiche come la conduzione di interventi chirurgici in remoto o lo streaming di film di qualità cinematografica a casa tua. L'altro percorso, su cui ci concentreremo qui, è ottimizzato per dispositivi a basso rendimento e bassa potenza come quelli nelle distribuzioni IoT.

## I DUE STANDARD

**LTE CAT M1** è definito come "una tecnologia per aree estese a bassa potenza che supporta l'IoT attraverso una minore complessità del dispositivo e fornisce una copertura estesa, consentendo nel contempo il riutilizzo della base installata LTE. Ciò consente una durata della batteria di almeno 10 anni per un'ampia gamma di casi d'uso, con costi del modem ridotti del 20-25 percento rispetto a quelli EGPRS". LTE-M ha una larghezza di banda più ampia, che ne consente l'utilizzo in più posizioni fisiche e **per applicazioni che richiedono un trasferimento di dati più elevato**. Tuttavia, questi vantaggi comportano un compromesso: **la flessibilità e la velocità di trasmissione dati lo rendono un'opzione più costosa**.

**NB-IoT** (abbreviato da NarrowBand per IoT) è definito come "una tecnologia **LPWA (Low Power Wide Area)** basata su standard sviluppata per abilitare una vasta gamma di nuovi dispositivi e servizi IoT". NB-IoT migliora significativamente il consumo energetico del dispositivo utente, la capacità del sistema e l'efficienza dello spettro, specialmente in una copertura profonda. La durata della batteria di oltre 10 anni può essere supportata per una vasta gamma di casi d'uso. **La sua banda più stretta e la minore frequenza di utilizzo dei dati sono dei vantaggi per l'efficienza e il costo della batteria, ma risulta anche meno flessibile** di LTE-M nei termini di dove e come può essere implementata.

**NB-IoT e CAT M1** (figura 1) **sono stati progettati per offrire** tecnologie affiliate, ma con **funzionalità complementari** adatte a due delle esigenze più comuni degli sviluppatori di prodotti: **un'opzione pone l'accento sull'estensione della durata della batteria e sulla riduzione dei costi e un'altra opzione offre maggiore flessibilità e prestazioni**. Vale la pena notare che CAT M1 ha ricevuto un notevole impulso nelle prime adozioni poiché i vettori lo hanno utilizzato per sostituire le reti 2G che erano in fase di eliminazione. Uno dei principali modi in cui 2G era ancora utilizzato era per le applicazioni di tracciamento e gestione della flotta, che sono essenzialmente applicazioni M2M ideali per le capacità di CAT M1. Ciò ha rappresentato una prima convalida di queste tecnologie e la motivazione a crescere per altre applicazioni IoT.

## 7 SUGGERIMENTI PER L'IMPLEMENTAZIONE

L'opzione cellulare presenta vantaggi significativi per molte applicazioni IoT, ma la complessità del suo utilizzo ha causato un avvio lento della sua adozione. Nordic Semiconductor ha recentemente riferito che "**l'adozione delle tecnologie IoT cellulari non viene rallentata dal prezzo elevato dei chipset, ma dalla complessità del loro design**". La complessità della progettazione con la tecnologia cellulare è amplificata in un ambiente IoT a causa di quanto sia difficile pianificare ed eseguire prodotti e implementazioni IoT di successo. Tutto

ciò cambierà presto, perché il vantaggio della notevole durata della batteria di LTE-M e NB-IoT, l'utilizzo dei dati economico e l'infrastruttura di celle onnipresenti saranno abbracciati da un numero crescente di produttori come protocollo ideale per molte delle iniziative IoT in cantiere .

Comprendere e razionalizzare il processo di progettazione sarà fondamentale per stare al passo con la crescente domanda del mercato di dispositivi a basso consumo e connessi in modalità wireless con la possibilità di connettersi alle reti LTE globali. Di seguito vengono descritti una serie di pratiche (figura 2) e avvertenze da tenere in conto nei progetti IoT basati su tecnologia cellulare.

## 1.LA GIUSTA TECNOLOGIA

**I due standard a basso consumo per l'IoT cellulare sono stati progettati per essere complementari e utilizzabili all'interno delle stesse implementazioni**. Alcuni dispositivi in una rete (e i compiti a cui sono assegnati) potrebbero richiedere durata della batteria maggiore e altre qualità per cui risulti ideale NB-IoT. Nel frattempo, un altro dispositivo altrove nella stessa rete potrebbe avere un ruolo che rende la flessibilità e la gamma di banda più ampia la soluzione migliore, ad esempio per fare un aggiornamento del firmware over-the-air. L'IoT cellulare non obbliga a prendere una decisione esclusiva. Per ogni dispositivo che viene progettato e per il ruolo che tale dispositivo dovrà svolgere può essere presa una decisione differente. Questo offre la responsabilità di fare la scelta giusta per ogni progetto e farlo pensando in anticipo a dove e come verrà utilizzato il dispositivo.

## 2.COMBINARE CON ALTRE TECNOLOGIE

Anche se si decide che NB-IoT e LTE-M sono le tecnologie giuste per il modo in cui le reti di sensori e i dispositivi inviano e ricevono dati sul cloud e sui sistemi informatici principali, ciò non significa che non ci sia un ruolo per altre tecnologie wireless. Ad esempio, la migliore strategia di comunicazione tra i dispositivi potrebbe essere quella di utilizzare **il Bluetooth a bassa potenza** in combinazione con LPWA cellulare, utilizzando entrambi sullo stesso dispositivo per ottimizzare il modo in cui i dispositivi comunicano, i ruoli che svolgono, le prestazioni della batteria e il trasferimento dei dati all'interno della rete. La possibilità di combinare le tecnologie dovrebbe essere affrontata all'inizio del processo di pianificazione di un progetto IoT, con la discussione su come scegliere la tecnologia giusta per ogni compito e quindi lavorare con moduli, modem e kit che semplificano il processo di collegamento e certificazione delle tecnologie su ogni dispositivo.

## 3.EVITARE L'OVERSHARING

Una delle cose più importanti da pianificare all'inizio dei progetti è il modo in cui la connettività cloud opera nell'implementazione dell'IoT. **La connettività cloud è fondamentale per ogni progetto IoT**, indipendentemente dalla tecnologia wireless utilizzata, ed è particolarmente importante per l'IoT cellulare poiché la gestione dei dati può avere un impatto elevato sui costi operativi. Il canale per il trasferimento di dati con questi protocolli IoT cellulari è piuttosto stretto, quindi la configurazione del dispositivo per l'invio continuo di dati al cloud comporterà costi significativi. In parole povere, l'oversharing può costare molto, soprattutto quando viene moltiplicato per dozzine o centinaia o migliaia di dispositivi IoT. Al contrario, la connettività cloud nelle distribuzioni IoT consente di eliminare dal dispositivo gran parte del carico di archiviazione e elaborazione dei dati al fine di ridurre al minimo il consumo di energia, le dimensioni, i costi, ecc. **C'è un equilibrio da trovare tra ciò che il dispositivo invia al cloud, cosa richiede il cloud e in che modo i due collaborano per ottimizzare prestazioni e costi dei dati**.

## 4.TEST E CERTIFICAZIONE

La certificazione per qualsiasi tecnologia wireless è difficile, come sa chi lavora con Bluetooth o Wi-Fi. Ma **la certificazione cellulare è molto più difficile perché ci sono tre livelli di certificazione**:

1. certificazione con le autorità nazionali di regolamentazione.
2. certificazione con associazioni industriali internazionali.
3. certificazioni con i singoli gestori che spesso hanno il proprio processo di certificazione prima che i dispositivi possano connettersi alle loro reti.

A causa di questo rigoroso e costoso processo di certificazione, è indispensabile pianificare la certificazione il primo giorno delle discussioni per un'iniziativa IoT, fattorizzando nel tempo le certificazioni nella sequenza temporale generale e cercando modi per semplificare le certificazioni utilizzando i moduli dei principali fornitori.

Un'altra cosa fondamentale a cui pensare all'inizio sono i test che ben si adattano alla combinazione di tecnologie che saranno presenti in molti prodotti IoT. La combinazione di più tecnologie (**BLE** più cellulari) in un singo-

lo dispositivo crea complessità quando si tratta di test, motivo per cui **è importante eseguire test co-ubicati con un fornitore che ha esperienza nelle sfumature di quel processo e può aiutare a completarlo con successo**. Meglio ancora se il partner di test può svolgere un ruolo consultivo in punti chiave del processo di progettazione per assicurare l'utilizzo delle migliori pratiche per la posizione dell'antenna, la schermatura e altre decisioni tecniche che incidono sui risultati dei test e sul successo della certificazione.

## 5.SCELTA DELL'ANTENNA

Il numero di antenne disponibili per i progetti wireless è sbalorditivo. Ce ne sono migliaia in ogni forma, dimensione e combinazione di tecnologie. L'antenna e il modo in cui viene integrata in un progetto hanno un impatto enorme sulle prestazioni di un dispositivo e sul successo dell'implementazione globale dell'IoT. La selezione dell'antenna diventa ancora più complessa per l'IoT cellulare perché occorre tener presente diversi fattori:

- la gamma di frequenze è ampia, con gestori diversi che utilizzano segmenti diversi della banda. Questo significa che l'identificazione dell'operatore gioca un ruolo nella scelta dell'antenna.
- la posizione fisica della distribuzione influisce sulla selezione dell'antenna.
- la combinazione di **NB-IoT e LTE-M** con una tecnologia wireless complementare come il Bluetooth influisce sulla scelta.

Lavorare con un consulente o un'azienda con esperienza nella selezione dell'antenna e nella progettazione di antenne personalizzate rende molto più probabile che l'antenna scelta per ciascun dispositivo sia ottimizzata per l'ambiente e l'applicazione.

## 6.L'IMPORTANZA DEL GESTORE

L'identificazione del gestore ha un impatto sostanziale sulle decisioni prese durante il processo di progettazione. Una discussione sui gestori dovrebbe svolgersi il più presto possibile nel processo di pianificazione di un'iniziativa IoT. Nella discussione, **è importante allineare le esigenze del progetto alle capacità di ciascun operatore telefonico**, scegliendone uno con la mappa di copertura, le prestazioni della rete, la certificazione e i costi più adatti al progetto.

## 7.STRATEGIA DI SOSPENSIONE

**Le opzioni della modalità di sospensione, che fanno parte dell'IoT cellulare, svolgono un ruolo fondamentale nel consentire la notevole durata della batteria** che si adatta perfettamente alle implementazioni dell'IoT. I termini chiave del ciclo di sospensione con cui familiarizzare sono **PSM** ed **eDRX** che operano insieme per rendere possibile la durata della batteria fino a 10 anni. **PSM (Power Saving Mode)**, che sta per modalità di risparmio energetico, **consente ai dispositivi di essere programmati per entrare in una modalità di sospensione profonda pur essendo raggiungibili**. È possibile programmare il ciclo di sospensione con una varietà di parametri. Il risparmio di energia equivale quasi a quello che si otterrebbe spegnendo un dispositivo, ma il dispositivo rimane ancora registrato nella rete e non ha bisogno di perdere tempo ed energia per ristabilire le connessioni. Ciò consente di risparmiare energia sia durante la sospensione che durante il periodo di riattivazione per inviare/ricevere dati o intraprendere un'azione. L'**eDRX (Extended Discontinuous Reception) consente di ottimizzare il periodo di tempo tra i cicli di paging**. I cicli di paging tipici per i dispositivi LTE sono lunghi 1,28 secondi. CAT M1 e NB-IoT consentono di espandere questi cicli di paging di un fattore 8 fino a 10,24 secondi, il che è significativo già da solo. Ma eDRX consente di programmare serie consecutive di periodi da 10,24 secondi prima che un dispositivo attivi effettivamente una finestra di paging. Ciò consente risparmi energetici maggiori consentendo di raggiungere l'obiettivo dei 10 anni nella durata della batteria .

## CONCLUSIONE

Come nella ricerca di indicazioni per una nuova destinazione è importante considerare attentamente tutti i percorsi, comprendere la tecnologia cellulare per l'IoT è essenziale al fine di effettuare la scelta più consapevole nei progetti IoT. Alcune strade portano più velocemente a destinazione, facendo risparmiare tempo e denaro, mentre altre potrebbero essere, nel peggiore dei casi, dei vicoli ciechi. Con questo articolo speriamo di aver fornito al lettore gli strumenti per leggere meglio la mappa delle possibilità e decidere la strada migliore da intraprendere nell'implementazione di progetti IoT utilizzanti tecnologia cellulare.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/semplificare-i-progetti-iot-basati-su-standard-cellulari-a-bassa-potenza

# AVR-IOT: UNA SCHEDA DI SVILUPPO PER COLLEGARE LE APPLICAZIONI IOT A GOOGLE CLOUD IN POCHI MINUTI

di **Andrea Garrapa**

*Progettare sistemi sicuri e connessi al cloud non deve essere un processo faticoso. Le schede di sviluppo AVR-IoT e PIC-IoT di Microchip offrono un punto di partenza perfetto per gli ingegneri che vogliono creare un qualsiasi dispositivi IoT, dai nodi sensori wireless ai sistemi di illuminazione intelligenti. Queste schede offrono una soluzione plug-and-play che combina un microcontrollore potente (AVR o PIC MCU), un elemento di sicurezza CryptoAuthentication e un modulo controller di rete Wi-Fi completamente certificato per fornire un modo semplice ed efficace per connettere le applicazioni integrate alla piattaforma Cloud IoT di Google. Queste schede di sviluppo sono la base ideale per la creazione senza sforzo di progetti collegati al cloud. In questo articolo ci focalizzeremo sulla scheda di sviluppo AVR-IoT WG.*

## INTRODUZIONE

Microchip e Google hanno stretto una partnership per fornire agli utenti uno strumento ideale per la realizzazione di progetti connessi al cloud. **Combinando potenti microcontrollori, un elemento di sicurezza come CryptoAuthentication e un controller di rete Wi-Fi completamente certificato, le schede di sviluppo AVR-IoT e PIC-IoT offrono il modo più semplice ed efficace per collegare le applicazioni incorporate alla piattaforma Google Cloud IoT**. Le schede consentono di effettuare una connessione rapida e diretta a Google Cloud, con account sandbox gratuito per visualizzare i dati di luce e temperatura. L'unica cosa da fare è scegliere l'architettura di microcontrollore preferita tra AVR o PIC per creare una connessione cloud facile e sicura.

Grazie ad un'enorme base di applicazioni embedded basate su AVR e PIC, la connessione dei progetti al cloud per il comando o il controllo remoto diventa semplice come non mai. **Caratteristiche principali delle schede di sviluppo AVR-IoT e PIC-IoT, sono le seguenti**:

- Passaggio da out-of-the-box a cloud-connected in 30 secondi
- MCU intelligente integrata offre potenza di elaborazione e semplicità
- Core Independent Peripherals (CIP) riducono ulteriormente il consumo energetico
- Autenticazione sicura con archiviazione della chiave privata basata su hardware
- Collegamento facilitato del proprio progetto anche qualora non si abbiano precedenti esperienze di progettazione wireless grazie al modulo Wi-Fi integrato
- Robusto ecosistema per generare rapidamente codice e sviluppare applicazioni.

**Le schede di sviluppo possono risultare ottimali nella progettazione di applicazioni nei seguenti settori:**

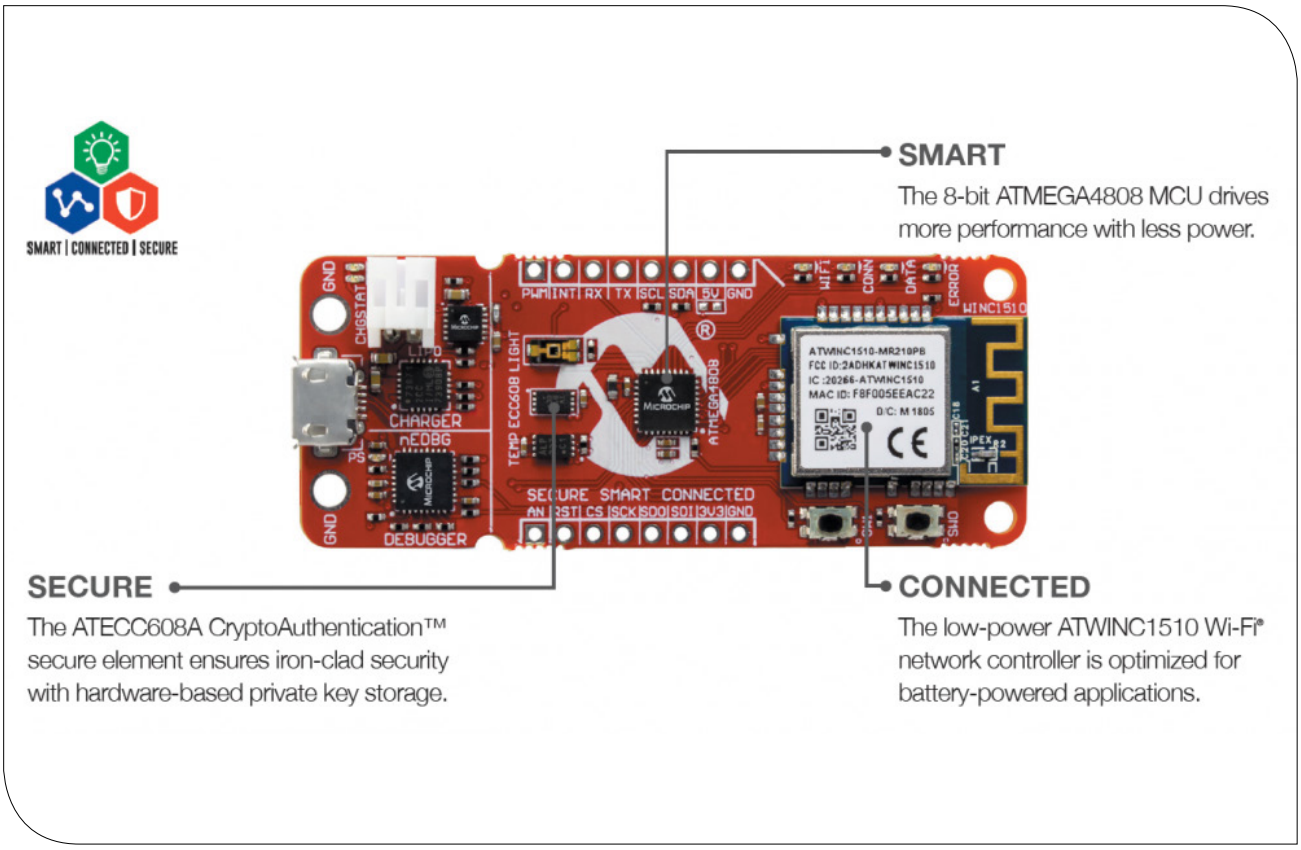



*Figura 1: la scheda di sviluppo AVR-IoT WG con evidenziati i tre componenti fondamentali.*

- Sensori e attuatori IoT Smart Home (illuminazione, controllo accessi, climatizzazione)
- Sensori industriali Smart City (qualità dell'aria, previsioni del traffico)
- Salute (pressione arteriosa, frequenza cardiaca)
- Sensori di controllo di processo Industry 4.0 (livello, pressione, temperatura, flusso)

## DIVIDERE IL PROBLEMA

Le schede sono state pensate per dimostrare che la progettazione di una tipica applicazione IoT può essere semplificata dividendo il problema in tre blocchi fondamentali:

### *Smart*

Il cuore di una scheda di sviluppo IoT risiede nel microcontrollore. La scelta tra AVR o PIC è puramente personale.
Queste MCU potenti ed efficienti consentono di portare in modo intelligente i dati sul Cloud. Tali microcontrollori sono scalabili e consentono di espandere le funzionalità IoT aggiungendo sensori personalizzati all'applicazione. Per la prototipazione rapida, le schede di sviluppo IoT sono supportate dagli IDE Studio 7 e MPLAB X e da strumenti di sviluppo grafico come START e MPLAB Code Configurator (MCC). Questi strumenti semplificano il collegamento di applicazioni esistenti al cloud o lo sviluppo di nuovi progetti IoT.

### *Connesso*

La connessione al Cloud è affidata all'ATWINC1510 di Microchip, un controller di rete Wi-Fi a banda singola intorno ai 2,4 GHz specificamente ottimizzato per applicazioni IoT a basso consumo.
Con un consumo energetico estremamente ridotto, la possibilità di archiviare vari certificati di sicurezza e 8 Mb di memoria flash integrata, questo dispositivo scarica tutte le attività di rete dalla CPU principale fornendo automaticamente una connessione sicura e l'autenticazione del server a Google Cloud.

### *Sicuro*

L'integrità di qualsiasi rete è determinata dal suo anello più debole. Poiché la superficie di attacco dei dispositivi IoT continua a crescere con una netta accelerazione, la sicurezza non può più essere tralasciata.
Le schede di sviluppo IoT affidano la crittografia dei dati

al più recente elemento di CryptoAuthentication basato su hardware del portafoglio di Microchip, ATECC608A. Utilizzando la memorizzazione delle chiavi ultra sicura, le contromisure crittografiche e l'oscuramento delle chiavi private da parte di utenti e software, questo elemento di sicurezza garantisce la massima fedeltà della trasmissione nel Cloud.

## LA SCHEDA DI SVILUPPO AVR-IOT WG

La scheda di sviluppo AVR-IoT WG (figura 1) è una piccola e facilmente espandibile piattaforma dimostrativa e di sviluppo per le soluzioni IoT, basata sull'architettura del microcontrollore AVR che utilizza la tecnologia Wi-Fi. Come già visto, i tre sottoproblemi per risolvere le applicazioni IoT connesse al Cloud, trovano soluzione in questa scheda di sviluppo nei seguenti elementi:

1. **Smart** - rappresentato dal microcontrollore **ATmega4808**
2. **Sicuro** - rappresentato dall'elemento di sicurezza **ATECC608A**
3. **Connesso** - rappresentato dal modulo controller Wi-Fi **WINC1510**

La scheda di sviluppo AVR-IoT WG presenta un **chip di interfaccia USB Nano Embedded Debugger (nEDBG)** che fornisce l'accesso a un'interfaccia seriale, un'interfaccia di archiviazione di massa per una facile programmazione "drag e drop", configurazione e accesso completo all'interfaccia UPDI del microcontrollore AVR per la programmazione e il debug direttamente

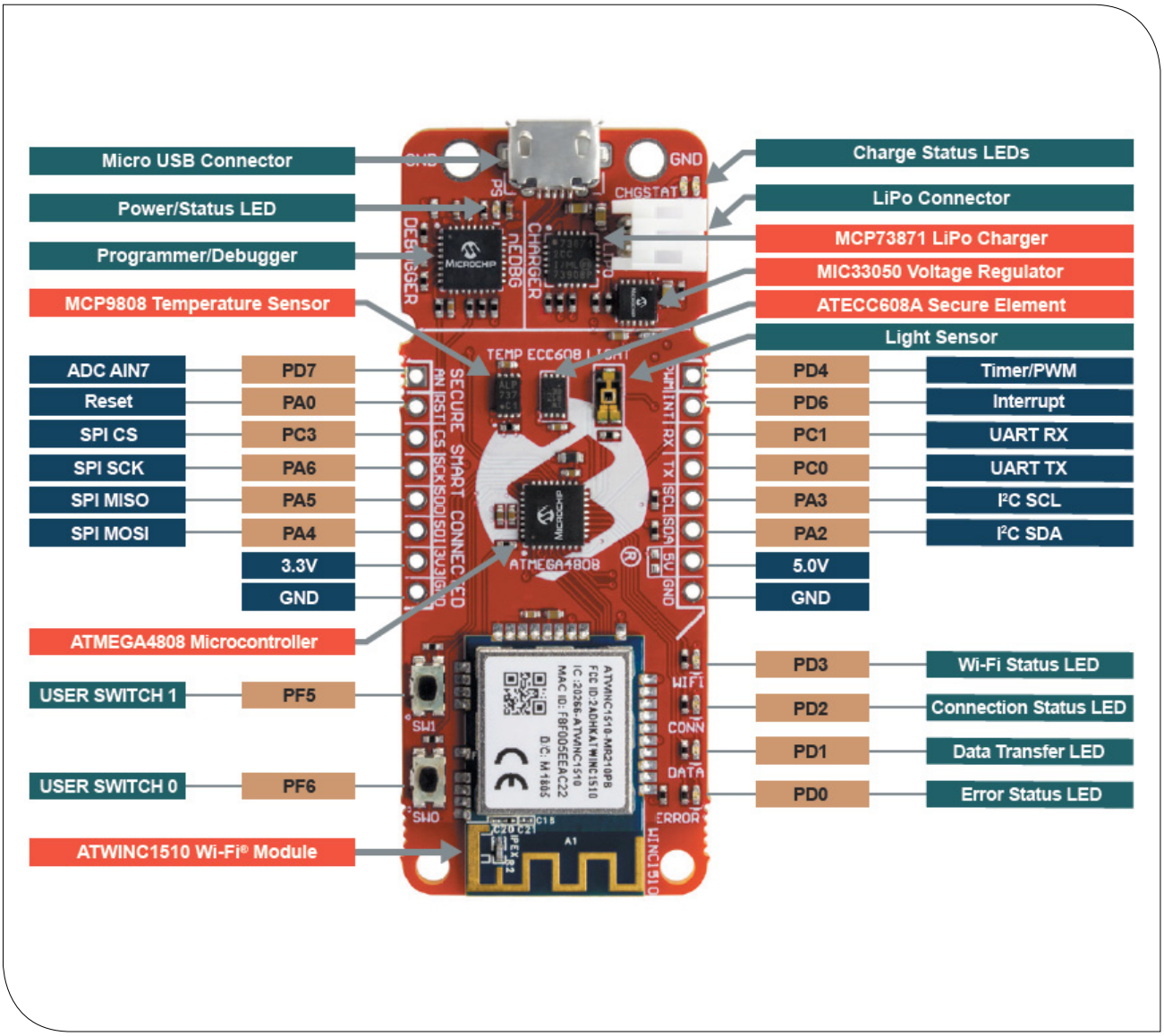



*Figura 2: piedinatura della scheda di sviluppo AVR-IoT WG.*

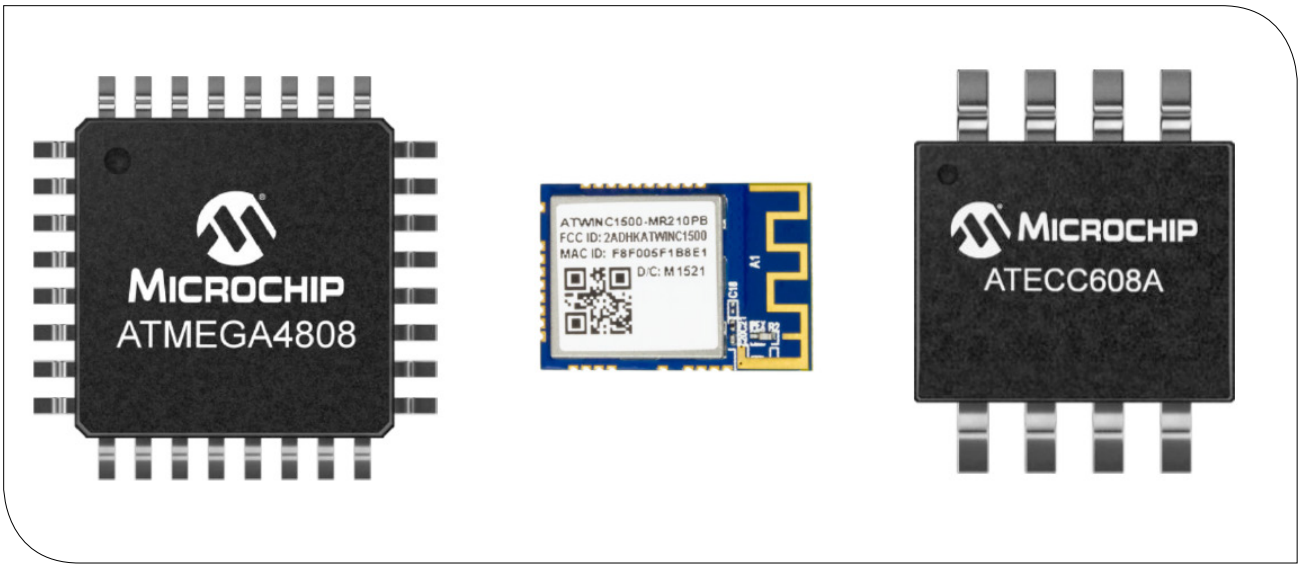



*Figura 3: partendo da destra la MCU ATmega4808, il modulo Wi-Fi WINC1510 e il modulo di sicurezza ATECC608A.*

da Microchip MPLAB X IDE e Atmel Studio 7.0 IDE. La scheda di sviluppo di AvR-IoT WG viene preprogrammata e configurata per dimostrare la connettività a Google Cloud IoT Core.

La scheda di sviluppo AVR-IoT WG presenta inoltre **due sensori**:

- Un sensore di luce
- Un sensore di temperatura ad alta precisione - MCP9808

Inoltre, **un connettore mikroBUS è fornito per espandere le capacità della scheda** con oltre 450 sensori e attuatori offerti da MikroElektronika attraverso un crescente portafoglio di schede Click.
**La scheda può essere cosi rapidamente trasformata in un rilevatore di movimento, un monitor della frequenza cardiaca o qualsiasi altra cosa si possa immaginare**.

## CARATTERISTICHE TECNICHE

Le caratteristiche tecniche principali del kit di sviluppo AVR-IoT WG vengono riassunte di seguito:

- Microcontrollore ATmega4808-MFR
- Quattro LED di stato
- Due pulsanti meccanici
- Modulo Wi-Fi WINC1510
- Sensore di luce TEMT6000
- Sensore di temperatura MCP9808
- Dispositivo CryptoAuthentication ATECC608A
- Connettore mikroBUS
- nEDBG
  - Identificazione della scheda in Atmel Studio e Microchip MPLAB X
  - Un LED verde per alimentazione e stato della scheda
  - Programmazione e debugging
  - Porta COM virtuale (CDC)
  - Due canali per analizzatore logico (DGI GPIO)
- Alimentato a batteria e USB
- Caricabatterie Li-Ion / LiPo
- Alimentazione fissa a 3.3V

La piedinatura del kit di sviluppo AVR-IoT WG viene illustrata nella figura 2.

## MICROCONTROLLORE ATMEGA4808

Il microcontrollore ATmega4808 (figura 3) appartiene alla serie-0 megaAVR, che abbiamo già trattato in altri articoli sul nostro sito.
Tale famiglia di microcontrollori estende le capacità in tempo reale dei sistemi di controllo combinando periferiche hardware intelligenti con le prestazioni a basso consumo del processore AVR. **Nelle applicazioni dove acquisizione analogica dei dati ed elaborazioni rappresentano le richieste essenziali, il microcontrollore ATmega4808 offre una conversione analogico digitale (ADC) ad alta velocità e Periferiche Indipendenti dal Processore (CPI) facili da configurare**.
Queste caratteristiche rendono il microcontrollore un compagno ideale in sistemi complessi basati su microcontrollori, o un eccellente processore indipendente in progetti per sistemi di controllo e comando.

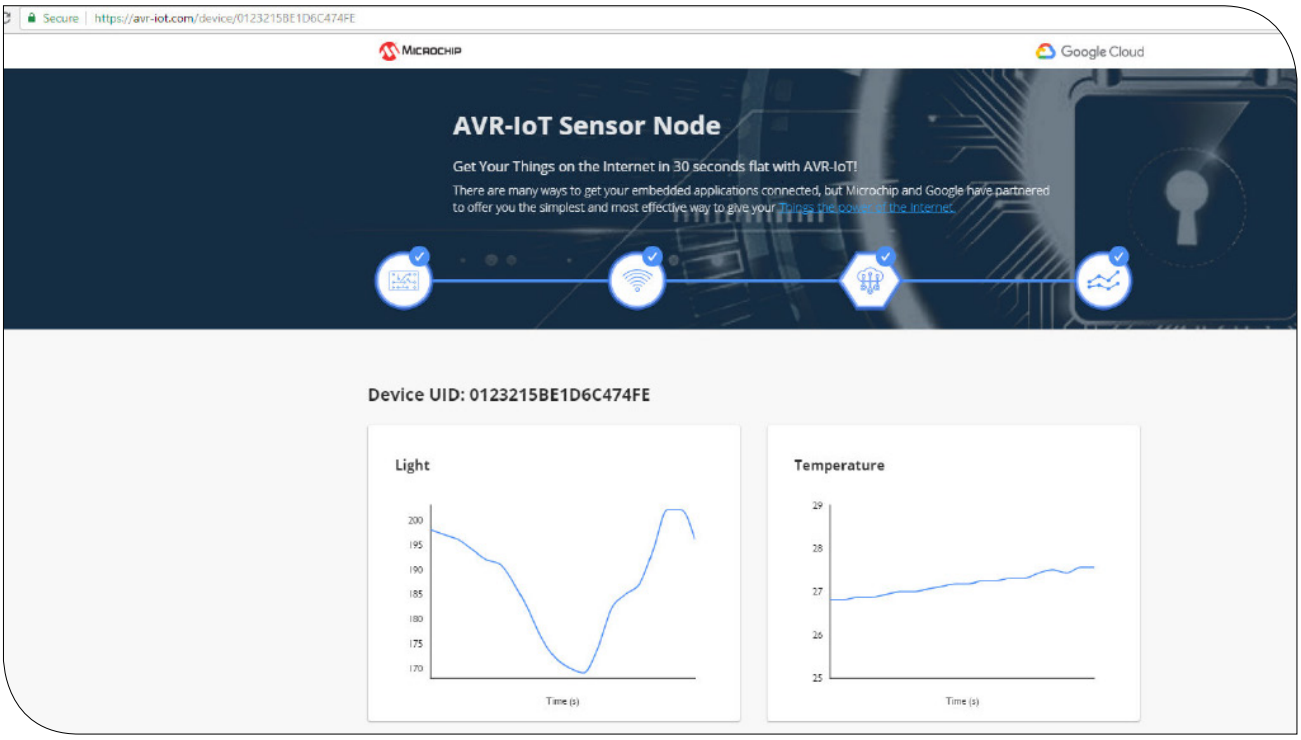



*Figura 4: schermata della pagina web avr-iot.com che mostra in tempo reale il grafico dei dati raccolti dai sensori integrati di luce e temperatura.*

## MODULO WI-FI WINC1510

**Il WINC1510 (figura 3) di Microchip è un modulo IoT 802.11 b/g/n a basso consumo, specificamente ottimizzato per applicazioni IoT a basso consumo**. Il modulo integra amplificatore di potenza (PA), amplificatore a basso rumore (LNA), interruttore, gestione dell'alimentazione e **un'antenna stampata o un connettore micro co-ax (u.FL) per un'antenna esterna con fattore di forma ridotto** (21,7 x 14,7 x 2,1 mm).

È interoperabile con i punti di accesso 802.11 b/g/n di vari fornitori. Questo modulo fornisce porte SPI per l'interfaccia con un controller host.

Il WINC1510 fornisce memoria flash interna e diverse interfacce periferiche, incluse UART e SPI. La sola fonte di clock esterna necessaria per il WINC1510 è l'oscillatore a cristallo integrato ad alta velocità (26 MHz).

L'interfaccia di comunicazione tra ATmega4808 e il modulo Wi-Fi WINC1510 è la SPI, insieme ad alcuni segnali di abilitazione e interrupt. Il resto delle connessioni sono lasciati scollegati.

## ATECC608A

ATECC608A (figura 3) è un elemento di sicurezza dal portafoglio CryptoAuthentication di Microchip con **funzionalità avanzate di Crittografia a curva ellittica (ECC)**.

Con ECDH e ECDSA integrati, questo dispositivo è ideale per il mercato in rapida crescita dell'IoT poichè copre l'intera gamma dei requisiti di sicurezza quali riservatezza, integrità dei dati e autenticazione per sistemi con MCU o MPU che eseguono algoritmi di crittografia/decrittografia. Analogamente a tutti i prodotti CryptoAuthentication di Microchip , il nuovo **ATECC608 impiega conservazione crittografata della chiave ultra sicura basata su hardware e contromisure crittografiche che eliminano qualsiasi potenziale intrusione legata a debolezze del software**.

Il dispositivo ATECC608A presente sul kit AVR-IoT WG viene utilizzato per archiviare la chiave pubblica e privata da usare nelle comunicazioni IoT sicure.

## ALTRE PERIFERICHE

La scheda AVR-IoT WG contiene un **Embedded Debugger (nEDBG)** per la programmazione on-board e il debug. nEDBG è un dispositivo USB composito di diverse interfacce: un debugger, un dispositivo di memorizzazione di massa, un gateway dati e una porta COM virtuale. Insieme ad Atmel Studio, l'interfaccia del debugger nEDBG può programmare ed eseguire il debug di ATmega4808.

La porta COM virtuale è collegata a una UART su ATmega4808 e fornisce un modo semplice per comunica-

re con l'applicazione di destinazione tramite terminale software.

Il **sensore di temperatura digitale MCP9808** converte le temperature tra -20 ° C e + 100 ° C in una parola digitale con precisione ± 0,25 ° C / ± 0,5 ° C (tipica / massima). Il sensore di temperatura MCP9808 è collegato all'ATmega4808 tramite I2C e un GPIO.

Il **sensore di luce TEMT6000X01** è montato sul kit AVR-IoT WG per misurare l'intensità della luce.

Il sensore è una fonte di corrente che induce una tensione sul resistore in serie, che a sua volta può essere misurata con l'ADC dell'ATmega4808.

La corrente è legata esponenzialmente all'illuminamento, da circa 10uA@20lx a 50uA@100lx. Il resistore di serie ha un valore di 10kΩ.

Sono disponibili quattro **LED** sulla scheda AVR-IoT WG che possono essere controllati con PWM o GPIO. I LED possono essere attivati portando la relativa linea di I/O a GND.

AVR-IoT WG contiene due **pulsanti meccanici**. Si tratta di pulsanti generici configurabili dall'utente. Quando si preme il tasto, la linea di I/O viene portata a GND.

## VISUALIZZARE DATI DEL CLOUD IN TEMPO REALE

Connettere la scheda al Cloud è davvero semplice e veloce, tant'è che gli sviluppatori di Microchip hanno stimato **il tempo tra lo spacchettamento della scheda e la connessione alla piattaforma di Google in circa 30 secondi**. Vediamo i passi da eseguire per effettuare una prima analisi dei dati raccolti dai sensori integrati di luminosità e temperatura grazie ai servizi di Google Cloud IoT.

La prima cosa da fare e collegare la scheda di sviluppo al PC con un cavo micro USB. Il personal computer riconoscerà la scheda come unità di archiviazione rimovibile denominata CURIOSITY.

L'unità CURIOSITY conterrà diversi file.

Cliccando su CLICK-ME.HTM verrà aperto il portale Google Cloud personale configurato in un browser. Immettendo le informazioni sulla rete wireless e cliccando su "Scarica configurazione" verrà scaricato il file di configurazione Wi-Fi, WIFI.CFG.

Trascinando e rilasciando il file scaricato WIFI.CFG nell'unità CURIOSITY, la scheda dovrebbe lampeggiare e connettersi automaticamente alla rete.

Una volta connessa la scheda, ci si potrà immediatamente immergere nell'account sandbox Google Cloud di Microchip.

**Tutte le schede di sviluppo AVR-IoT sono preregistrate nell'account sandbox Google Cloud di Microchip. Questo account è impostato a solo scopo dimostrativo.**

**Tutti i dati raccolti dai sensori delle schede di sviluppo AVR-IoT sono pubblicate sull'account sandbox di Microchip**.

Non vi è alcuna memorizzazione permanente o raccolta dei dati pubblicati dalle schede collegate tramite account sandbox Microchip.

**L'archiviazione completa delle funzionalità di Google Cloud sarà disponibile per l'utente dopo aver rimosso la scheda dall'ambiente demo e migrato su un account privato**.

Una volta che la scheda è connessa al Wi-Fi e al Cloud, la pagina web avr-iot.com mostrerà in tempo reale il grafico dei dati raccolti dai sensori di luce e temperatura integrati (figura 4).

I dati vengono trasferiti e trasformati dal sensore al cloud attraverso un oggetto JSON: una stringa ASCII formattata come segue: {'Light': XXX, 'Temp': YYY}, dove XXX e YYY sono valori numerici espressi in notazione decimale.

## CONCLUSIONE

Il servizio Cloud IoT di Google è uno strumento che consente di archiviare, elaborare, analizzare i dati provenienti da milioni di dispositivi.

Michochip con le sue schede di sviluppo AVR-IoT WG e PIC-IoT WG offre degli strumenti potenti per collegare in maniera rapida e sicura le applicazioni integrate con la piattaforma Cloud di Google.

Nella progettazione di queste schede, Microchip ha posto l'enfasi su tre aspetti fondamentali: la potenza, rappresentata dal microcontrollore ATmega4808; la sicurezza rappresentata dal modulo ATECC608A; e la connettività rappresentata dal controller Wi-Fi WINC1510.

Insieme ad altre periferiche integrate, le schede di sviluppo si pongono come un ambiente ideale per realizzare i propri progetti IoT e collegarli rapidamente e facilmente alla piattaforma Cloud di Google per usufruire dei servizi offerti.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/avr-iot-una-scheda-di-sviluppo-per-collegare-le-applicazioni-iot-a-google-cloud-in-pochi-minuti

# SDK PER ESPERTINO: GUIDA ALL'INSTALLAZIONE

di **Stefano Lovati**

*ESPertino è una scheda di prototipazione rapida particolarmente orientata alle applicazioni IoT. Sappiamo che tra i suoi punti di forza risiede la possibilità di essere programmata tramite l'ambiente Arduino IDE, uno strumento di semplice e immediato utilizzo anche per i non specialisti. In questo articolo ci occuperemo dell'ambiente di sviluppo ESP-IDF, l'SDK ufficiale per tutte le schede basate sul modulo ESP32. Anche se meno semplice rispetto ad Arduino IDE, l'SDK permette di sfruttare efficacemente le risorse hardware e software del modulo ESP32, consentendo il pieno controllo delle funzionalità offerte dalla scheda ESPertino.*

## INTRODUZIONE

Quando si utilizza una scheda di **prototipazione rapida** potente e versatile come ESPertino, l'approccio più comunemente seguito è quello di arrivare nel più breve tempo possibile a un prototipo dell'applicazione funzionante, qualcosa che possa essere utilizzata come "demo" del prodotto finale. In questo processo l'**ambiente di sviluppo** utilizzato esercita un ruolo fondamentale, ottimizzando e riducendo i tempi di sviluppo dell'applicazione. L'ambiente Arduino IDE, opportunamente esteso per supportare il modulo ESP32, è sicuramente in grado di soddisfare pienamente questi requisiti. In realtà, Arduino IDE e il suo plugin per ESP32 rappresentano un'astrazione (una "virtualizzazione" come potremmo dire oggi) di un sistema che in realtà (dietro le quinte e in modo del ltutto trasparente rispetto all'utente) viene programmato utilizzando strumenti software come l'SDK e il kernel FreeRTOS. Per meglio comprendere questa "stratificazione" dei livelli software disponibili sul modulo ESP32 possiamo fare riferimento alla **Figura 1**.

Il livello più basso della gerarchia è rappresentato dall'**hardware**, ovvero il modulo ESP32 equipaggiato con un microcontrollore Tensilica Xtensa LX6 nella versione dual core. Oltre alla MCU e alle sue periferiche, questo livello include alcune funzioni mappate direttamente dal produttore nella memoria ROM: questo codice gestisce funzionalità di basso livello del modulo (ad esempio gli stack Bluetooth e Ethernet) e viene invocato direttamente dagli strati superiori (in particolare l'SDK). Immediatamente sopra l'hardware "poggia" il sistema operativo, più precisamente il kernel **FreeRTOS** adattato per gestire l'architettura a doppio core dell'LX6. Il kernel fornisce il supporto per tutte le operazioni relative alla creazione, cancellazione, sincronizzazione e comunicazione tra task. Il kernel non fornisce invece alcun supporto per la gestione delle periferiche (SPI, I2C, I2S, DAC/ADC, GPIO e altro ancora): questo compito è a carico dell'**SDK**. L'SDK mette infatti a disposizione dello

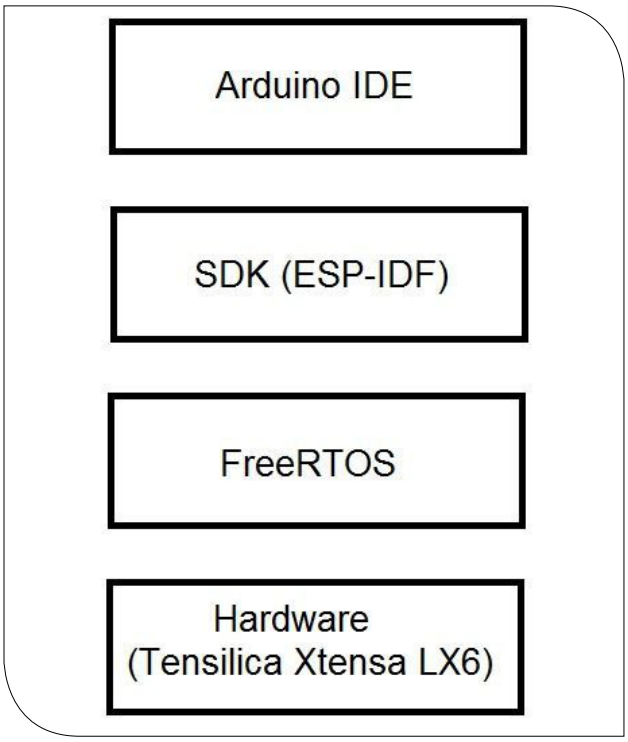



*Figura 1: i differenti livelli software presenti nel modulo ESP32*

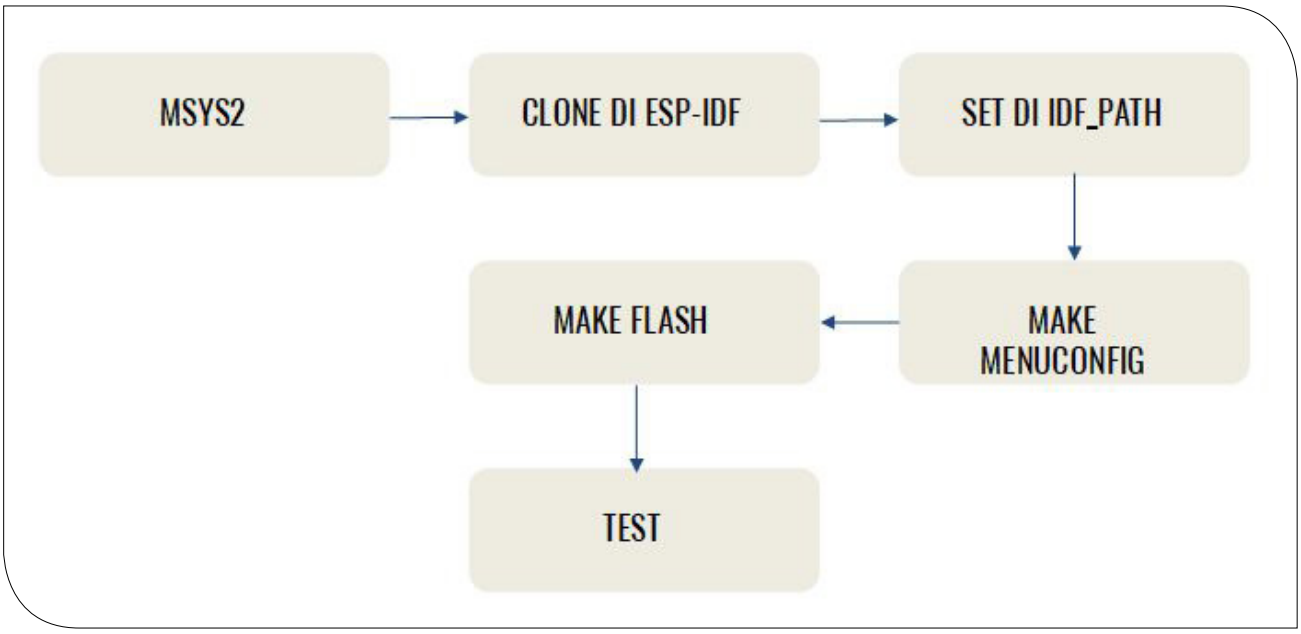



*Figura 2: schema a blocchi con i passi necessari per l'installazione dell'SDK sotto Windows*

sviluppatore tutti i driver (che spesso includono la creazione di uno o più task FreeRTOS) per la gestione di alto livello delle periferiche, incluso il Bluetooth BLE e il WiFi. Le chiamate alle funzioni di basso livello in ROM sono dunque "annegate" nell'SDK e lo sviluppatore non deve preoccuparsene: è sufficiente fare riferimento alla documentazione dell'SDK, in cui sono riportate tutte le API, i tipi e le costanti da utilizzare a livello applicativo. Arduino IDE occupa il livello più alto della gerarchia software, fornendo la massima astrazione rispetto sia all'hardware sia ai livelli software sottostanti. Utilizzando ad esempio le librerie di base dell'ambiente (come ad esempio la libreria Serial) l'utente non trova alcuna differenza, a livello di programmazione dello sketch, tra un sistema hardware basato su un microcontrollore AVR e una board basata sul modulo ESP32.

L'SDK **ESP-IDF**, anche se meno "user friendly" rispetto ad Arduino IDE, merita comunque una certa attenzione e in alcuni casi si dimostra insostituibile. E' ad esempio possibile gestire alcune operazioni non altrimenti eseguibili come:

- gestione delle partizioni sulla memoria flash: si può decidere tra destinare l'intera memoria flash all'applicazione, oppure dividerla in due segmenti in modo tale da gestire l'aggiornamento del programma applicativo con la tecnica **OTA** (aggiornamento "live", senza richiesta di passaggio per il bootloader);
- gestione di funzionalità avanzate del modulo ESP32, come ad esempio la modalità a bassissimo assorbimento (**deep sleep**);
- possibilità di modificare e ricompilare la **libreria FreeRTOS** in modo da personalizzare il comportamento del kernel (Arduino IDE utilizza tale libreria ma dal suo ambiente non è possibile rigenerarla);
- possibilità di personalizzare i parametri di configurazione e le funzionalità di un'applicazione tramite il comando **make menuconfig** (di cui parleremo più avanti).

In **Figura 2** possiamo osservare lo schema a blocchi delle operazioni necessarie per eseguire l'installazione dell'SDK. Per semplicità faremo riferimento all'installazione in ambiente Windows (significativa perché richiede l'utilizzo di tool non disponibili in modo nativo su tale sistema), ma ricordiamo che l'SDK è disponibile anche per i sistemi Linux e Mac OS.

Vediamo ora in dettaglio i vari passi su cui si articola l'installazione dell'SDK per il modulo ESP32, tenendo presente che tutto il software e la relativa documentazione sono disponibili su GitHub a questo indirizzo.

## INSTALLAZIONE DI MSYS2

Il sistema operativo Windows non dispone di un ambiente di compilazione integrato basato su "make", per cui dovremo anzitutto installare un ambiente compatibile con la linea di sviluppo GNU. La soluzione più immediata è quella di utilizzare l'ambiente MSYS2, di cui è disponibile una versione adatta allo sviluppo per il modulo ESP32 e corredata degli strumenti Git e Python, entrambi richiesti dalla linea di sviluppo. A tale scopo è

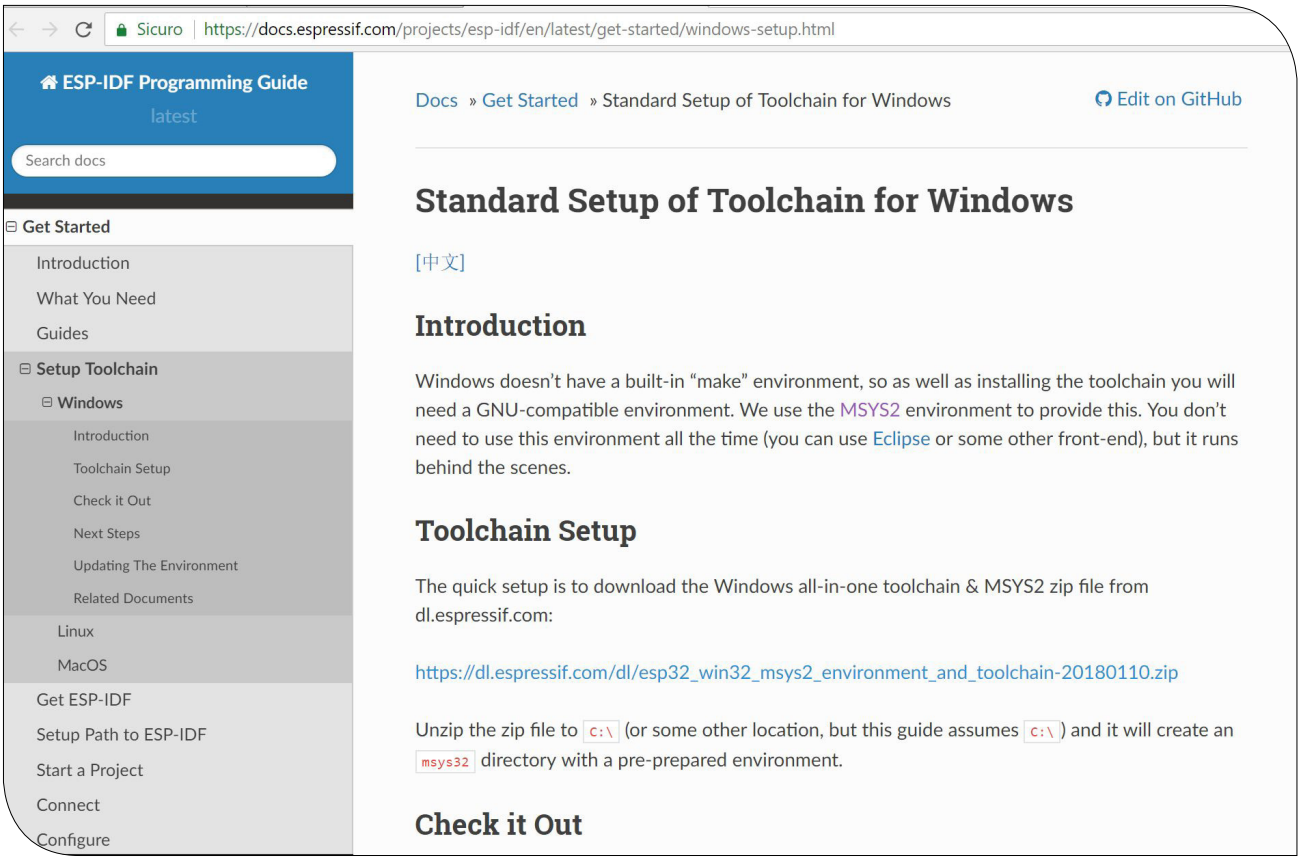



*Figura 3: download del pacchetto MSYS2*

sufficiente accedere a questa pagina web e cliccare sul link alla versione più recente di MSYS2 preparata da Espressif Systems, come indicato in **Figura 3**.
Una volta scaricato lo zip, è sufficiente decomprimerlo in una cartella sul proprio hard disk: il nostro consiglio è quello di eseguire l'installazione direttamente nella root del disco C (C:\), ciò semplificherà notevolmente le successive fasi di configurazione dell'ambiente. Dopo la decompressione del file, l'ambiente sarà disponibile nella cartella C:\msys32. Possiamo a questo punto verificare il corretto comportamento dell'ambiente attivando il programma C:\msys32\mingw32.exe: avremo a questo punto disponibile una bash shell che utilizzeremo per interfacciarci con l'ambiente di sviluppo ESP-IDF (**Figura 4**).

## AMBIENTE ESP-IDF

L'ambiente ESP-IDF, disponibile su GitHub a questo indirizzo, deve essere copiato (o meglio, "clonato" come si dice nella terminologia di Git) sulla propria macchina nella cartella esp. Apriamo dunque una shell mingw32.exe e digitiamo i seguenti comandi:

```
mkdir -p ~/esp
cd ~/esp
```

Il primo comando serve a creare la cartella esp a partire dalla propria home directory (~/), mentre il secondo comando serve a cambiare la directory corrente in esp. Possiamo a questo punto clonare sulla nostra macchina il repository remoto contenente l'SDK tramite il comando:

```
git clone --recursive https://github.com/espressif/esp-idf.git
```

Si noti come l'indirizzo del repository possa essere ottenuto facendo il cut & paste dalla pagina GitHub, come indicato in **Figura 5** (il paste nella shell mingw32.exe si esegue cliccando il tasto destro e selezionando l'omonima opzione dal menu contestuale).
L'output prodotto dal lancio di questo comando (il cui completamento può richiedere un certo tempo, dipendente anche dalla qualità della connessione internet utilizzata) è visibile in **Figura 6**. Al termine dell'operazione, l'SDK sarà disponibile nella cartella ~/esp/esp-idf.

## IMPOSTAZIONE DELLA VARIABILE PATH

Con riferimento alla **Figura 2**, il passo successivo consiste nella configurazione della variabile di ambiente

```
Copying skeleton files.
These files are for the users to personalise their msys2 experience.

They will never be overwritten nor automatically updated.

'./.bashrc' -> '/home/LOVATI//.bashrc'
'./.bash_logout' -> '/home/LOVATI//.bash_logout'
'./.bash_profile' -> '/home/LOVATI//.bash_profile'
'./.inputrc' -> '/home/LOVATI//.inputrc'
'./.profile' -> '/home/LOVATI//.profile'
'./.vimrc' -> '/home/LOVATI//.vimrc'

LOVATI@NB-LOVATI MINGW32 ~
$
```

*Figura 4: attivando mingw32.exe si ha disposizione una bash shell completa*

contenente il percorso (path) all'SDK appena installato. Questa variabile è fondamentale, in quanto è richiesta dall'ambiente di compilazione di una qualunque applicazione (tramite il comando make oppure tramite un ambiente IDE, come ad esempio Eclipse). La variabile d'ambiente (il cui nome deve essere **IDF_PATH**) può essere configurata in modo dinamico ad ogni apertura della shell mingw32.exe, oppure in modo statico inserendola in uno script avviato ad ogni attivazione della shell (noi sceglieremo quest'ultima opzione, in quanto ci sembra quella più semplice e pratica). Dovremo anzitutto creare un file script, che possiamo chiamare export_idf_path.sh, all'interno della cartella C:/msys32/etc/profile.d/. Questa cartella contiene tutti i file script che vengono automaticamente caricati ed eseguiti ogni volta che si apre una shell MSYS2. Possiamo utilizzare un editor qualunque (anche Notepad di Windows) e creare il file direttamente nella stessa cartella oppure copiarlo tramite il file manager. Il file deve contenere la seguente (ed unica) riga:

```
export IDF_PATH="C:/msys32/home/user-name/esp/esp-idf"
```

dove al posto di "user-name" occorre inserire il proprio nome utente, visibile nel prompt della shell MSYS2. Dopo aver salvato (oppure copiato) il file export_idf_path.sh nel percorso indicato in precedenza, occorre chiudere e riaprire la shell MSYS2. Per verificare che la variabile di ambiente sia stata correttamente impostata, digitare nella shell il comando indicato in **Figura 7**.

## CONFIGURAZIONE DI UN ESEMPIO

L'ambiente di sviluppo ESP-IDF include già due applicazioni di esempio, contenute all'interno della cartella ~/esp/esp-idf/examples/get-started. Utilizzeremo uno di questi due esempi (hello_world) per prendere piena confidenza con l'ambiente, configurando, compilando e scaricando sulla scheda ESPertino l'applicazione stessa. Utilizziamo a questo scopo un'approccio di tipo conservativo, copiando l'intera cartella dell'esempio in un percorso separato, in modo tale da poter agire in completa libertà, senza timore di alterare in modo lesivo i file originari (peraltro recuperabili in ogni momento tramite accesso al repository GitHub). Procediamo quindi copiando l'esempio hello_world nella cartella ~/esp tramite i seguenti comandi:

```
cd ~/esp
cp -r $IDF_PATH/examples/get-started/hello_world .
```

Siamo a questo punto pronti per eseguire la configurazione dell'applicazione di esempio. Ci occorre ora la board ESPertino, che andrà collegata a una porta USB

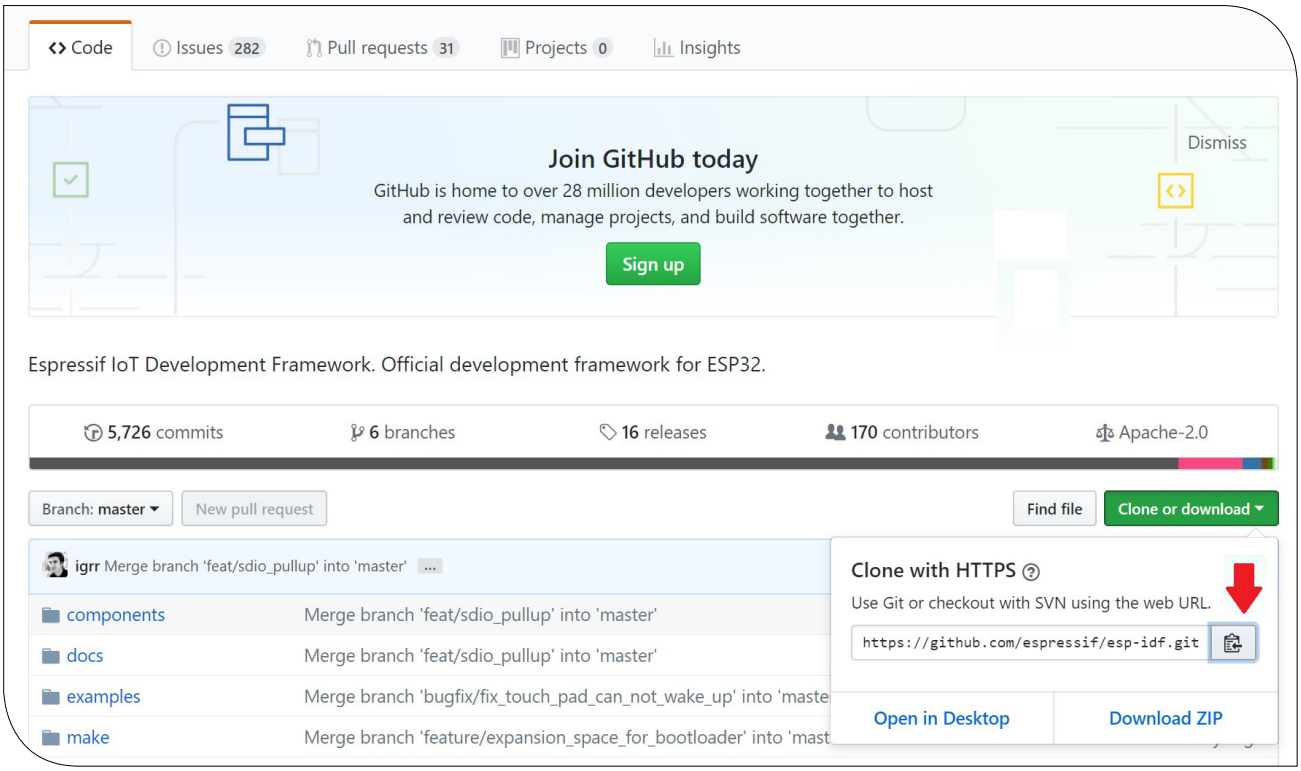



*Figura 5: premendo sull'icona a destra, si esegue la copia nello scratchpad*

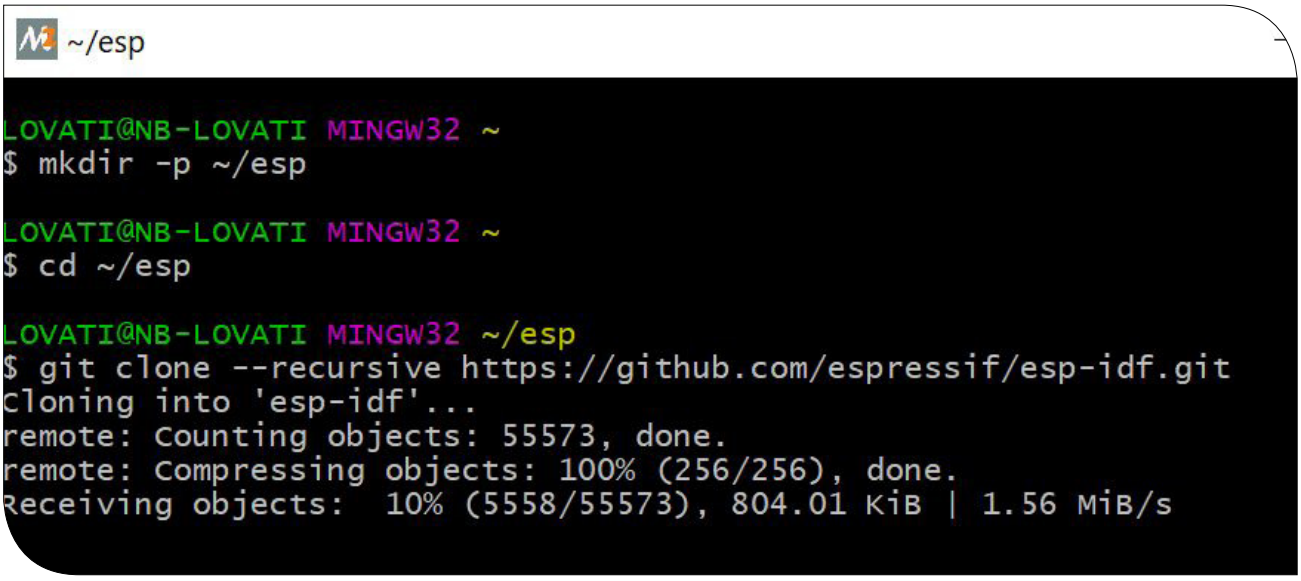



*Figura 6: lancio del comando di clonazione del repository Git dell'SDK*

del nostro computer, prendendo nota della porta seriale (COMxx) allocata per la comunicazione con il driver del convertitore USB-seriale. Un esempio è mostrato in **Figura 8**, in cui il driver Silicon Labs è stato associato alla porta seriale COM6. Alternativamente, si può utilizzare un emulatore di terminale come TeraTerm, il quale provvede automaticamente a presentare solo le porte seriali riconosciute dal sistema operativo.

Dobbiamo ora accedere alla cartella contenente l'applicazione hello_world e lanciare la configurazione del programma tramite i seguenti comandi impartiti dalla shell MSYS2 (mingw32.exe):

```
cd ~/esp/hello_world
make menuconfig
```

Dopo un'attesa di qualche secondo, al posto della shell

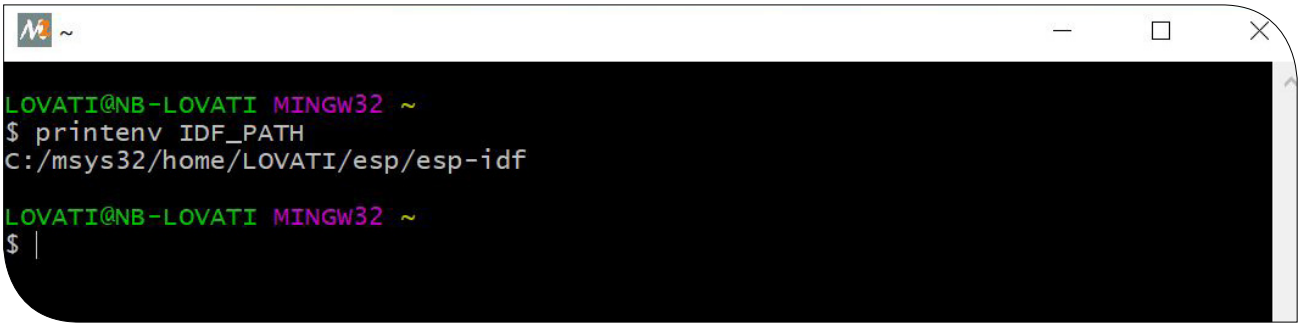



*Figura 7: verifica della corretta impostazione della variaibile IDF_PATH*

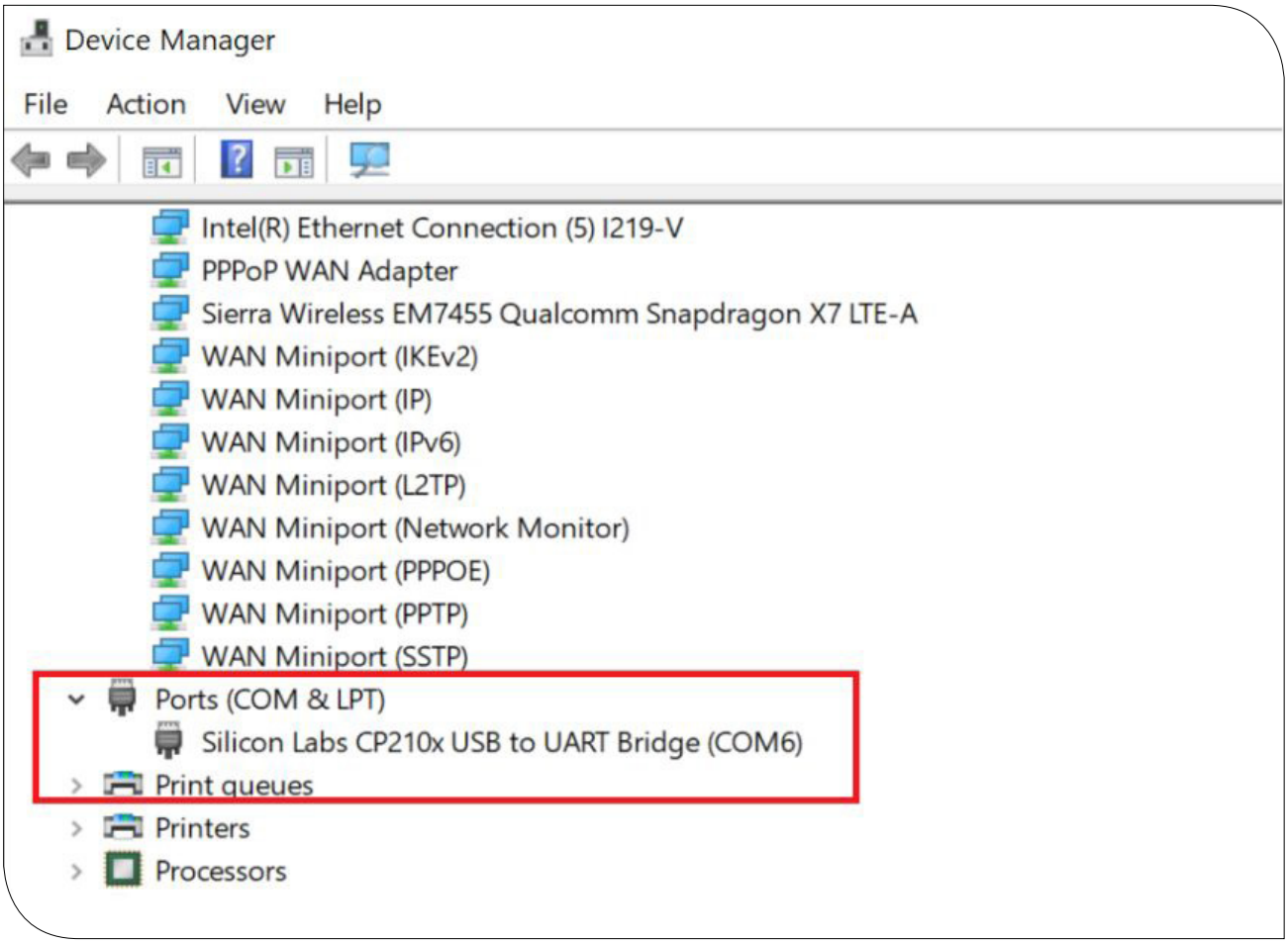



*Figura 8: come individuare la porta seriale utilizzata dalla scheda ESPertino*

vedremo l'immagine visualizzata in **Figura 9**. Si tratta in sostanza del menu di configurazione di ogni applicazione sviluppata con l'SDK, in cui si possono selezionare, prima della compilazione, diverse opzioni. Una volta modificata e salvata la configurazione, l'SDK genera automaticamente un file di configurazione (in pratica un file header, quindi con estensione .h) che viene compilato con il resto dell'applicativo.

Relativamente a questo programma di esempio, molto semplice, ci interessa solamente configurare la porta seriale utilizzata per la comunicazione con la scheda, annotata in precedenza. Selezioniamo quindi dal menu la voce "serial flasher config" e premiamo Enter. Ci verrà proposta una nuova schermata (**Figura 10**), in cui dovremo selezionare e modificare il contenuto della prima riga inserendo la porta seriale di interesse (ad esempio COM6), come indicato nelle **Figure 10** e **11**.

Salviamo quindi il file di configurazione e usciamo dal menu selezionando la voce Exit (attenzione, l'operazione può richiedere un certo tempo, durante il quale il prompt della shell non è disponibile).

Vale la pena proporre qualche semplice indicazione e

```
~/esp/hello_world
/home/LOVATI/esp/hello_world/sdkconfig - Espressif IoT Development Framework C
                Espressif IoT Development Framework Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

          SDK tool configuration  --->
          Bootloader config  --->
          Security features  --->
          Serial flasher config  --->
          Partition Table  --->
          Compiler options  --->
          Component config  --->

        <Select>    < Exit >    < Help >    < Save >    < Load >
```

*Figura 9: il menu di configurazione di ogni applicazione sviluppata con l'SDK*

```
~/esp/hello_world
/home/LOVATI/esp/hello_world/sdkconfig - Espressif IoT Development Framework C
n→ Serial flasher config
                Serial flasher config
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

      (/dev/ttyUSB0) Default serial port
          Default baud rate (115200 baud)  --->
      [*] Use compressed upload
          Flash SPI mode (DIO)  --->
          Flash SPI speed (40 MHz)  --->
          Flash size (2 MB)  --->
      [*] Detect flash size when flashing bootloader
          Before flashing (Reset to bootloader)  --->
          After flashing (Reset after flashing)  --->
          'make monitor' baud rate (115200 bps)  --->

        <Select>    < Exit >    < Help >    < Save >    < Load >
```

*Figura 10: configurazione della porta seriale con il valore di default*

consiglio relativi all'utilizzo del menu di configurazione (menuconfig):

- per navigare tra le opzioni dei menu utilizzare i tasti cursore freccia verso l'alto e freccia verso il basso;
- premere il tasto Enter per accedere a un sotto-menu, oppure il tasto Esc per uscire da un sotto-menu;
- premere il tasto ? per visualizzare la schermata di aiuto (dalla quale si esce premendo il tasto

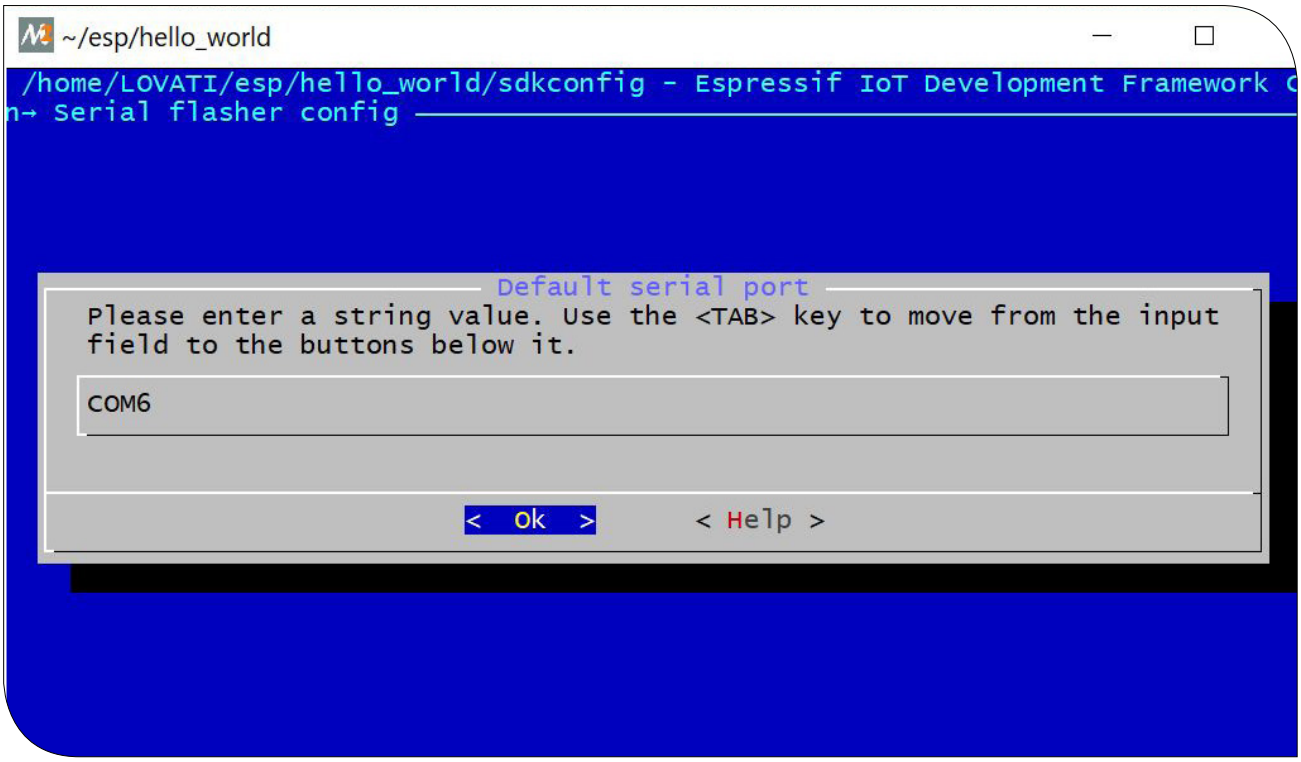



*Figura 11: modifica del campo con inserimento del valore corretto*

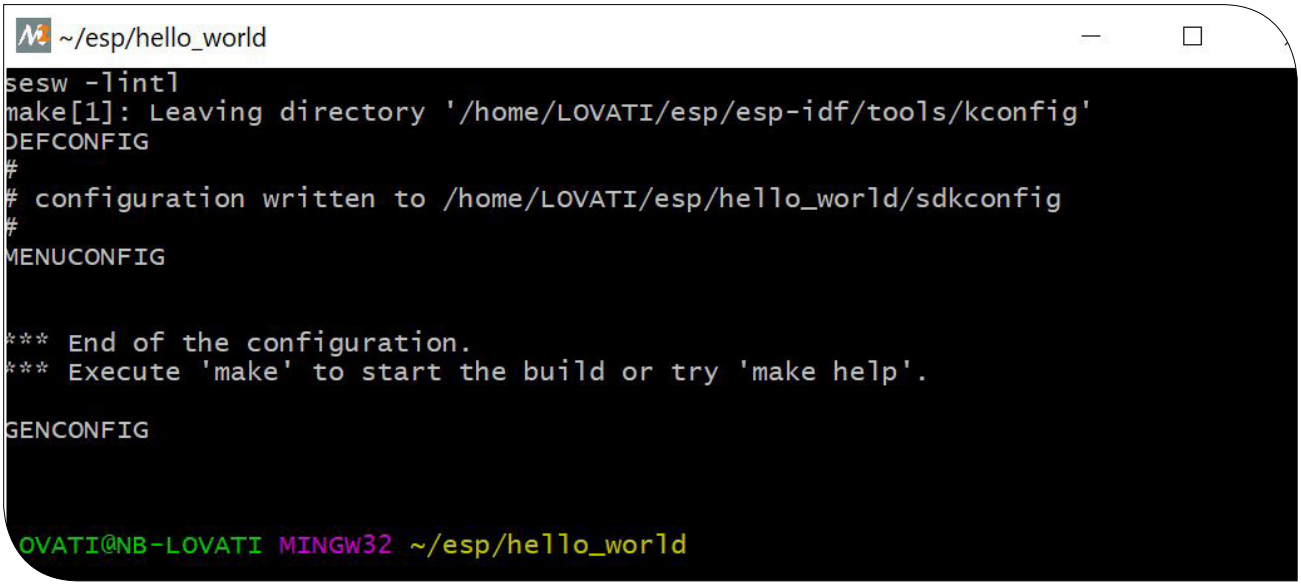



*Figura 12: log prodotto dopo il completamento del comando make menuconfig*

Enter);

- utilizzare la barra spaziatrice oppure i tasti Y (yes) e N (no) rispettivamente per abilitare o disabilitare le voci di menu contraddistinte da una checkbox "[*]";
- premendo il tasto ? quando è visualizzata una voce di configurazione, viene visualizzato l'help relativo all'opzione medesima;
- premere il tasto / per eseguire la ricerca di una voce di menu.

## COMPILAZIONE

Sempre con riferimento allo schema a blocchi di **Figura 2**, siamo ora giunti al blocco con la dicitura "Make Flash" e pertanto siamo pronti a lanciare la compilazione del programma di esempio.

Se osserviamo l'output prodotto sulla shell MSYS2 dopo l'esecuzione del comando make menuconfig (**Fi-**

```
~/esp/hello_world

LOVATI@NB-LOVATI MINGW32 ~/esp/hello_world
$ make flash
Flashing binaries to serial port COM6 (app at offset 0x10000 )...
esptool.py v2.3.1
Connecting....
Chip is ESP32D0WDQ6 (revision 0)
Features: WiFi, BT, Dual Core
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 20944 bytes to 12370...
Wrote 20944 bytes (12370 compressed) at 0x00001000 in 1.1 seconds (effective 152.5 kbit/s)...
Hash of data verified.
Compressed 139424 bytes to 69411...
Wrote 139424 bytes (69411 compressed) at 0x00010000 in 6.2 seconds (effective 180.3 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.0 seconds (effective 1261.2 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

LOVATI@NB-LOVATI MINGW32 ~/esp/hello_world
```

*Figura 13: output prodotto dal comando make flash*

```
~/esp/hello_world

I (194) cpu_start: App cpu up.
I (213) heap_init: Initializing. RAM available for dynamic allocation:
I (220) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (226) heap_init: At 3FFB3300 len 0002CD00 (179 KiB): DRAM
I (232) heap_init: At 3FFE0440 len 00003BC0 (14 KiB): D/IRAM
I (238) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (245) heap_init: At 40088A9C len 00017564 (93 KiB): IRAM
I (251) cpu_start: Pro cpu start user code
I (269) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
Hello world!
This is ESP32 chip with 2 CPU cores, WiFi/BT/BLE, silicon revision 0, 4MB extern
al flash
Restarting in 10 seconds...
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
Restarting in 6 seconds...
Restarting in 5 seconds...
Restarting in 4 seconds...
Restarting in 3 seconds...
Restarting in 2 seconds...
```

*Figura 14: l'utilità di monitor seriale*

**gura 12**), notiamo che ci viene suggerito di lanciare il comando make per eseguire la compilazione, oppure make help per ottenere tutte le informazioni relative al comando.

L'esecuzione del comando make produrrà la compilazione di tutti i file richiesti dall'applicazione (incluse le librerie dell'SDK) e produrrà in uscita il file binario pronto per essere scaricato su ESPertino. Poiché è la prima volta che compiliamo l'SDK dopo averlo installato sulla nostra macchina, dovremo generare tutte le librerie necessarie e l'operazione richiederà alcuni minuti (a seconda delle prestazioni del PC utilizzato).

Le successive compilazioni (a condizione che abbiano impatto solo sui file dell'esempio e non su quelli di siste-

ma) procederanno invece in modo più spedito. Lanciando il comando make con l'opzione flash (make flash) si può eseguire con un solo comando prima la compilazione e successivamente l'upload del file binario sulla board ESPertino (quest'ultima dovrà essere collegata alla porta seriale configurata con il comando make menuconfig).
Eseguiamo quindi il build dell'applicazione impartendo il seguente comando dalla shell MSYS2:

```
make flash
```

L'effetto di questo comando è quello di compilare i file sorgenti che compongono l'applicazione e le componenti dell'SDK, generare il bootloader, la tabella delle partizioni (partition table) e il file binario dell'applicativo, procedendo poi con la programmazione degli stessi sulla scheda ESPertino.
In **Figura 13** possiamo osservare l'output prodotto dal processo di compilazione e successivo upload del file binario sulla scheda ESPertino.

## VERIFICA E TEST

L'applicazione che abbiamo scelto come esempio è molto semplice: dopo aver acquisito alcune informazioni relative alle caratteristiche hardware, le visualizza sulla linea seriale. Viene quindi fatto partire un timeout di 10 secondi allo scadere del quale la board si resetta e il ciclo si ripete nuovamente.
Per visualizzare l'output prodotto dall'esecuzione dell'applicazione possiamo utilizzare un comune programma di emulazione del terminale (come ad esempio TeraTerm), configurato alla velocità di 115200 bps. L'ambiente di sviluppo ESP-IDF, tuttavia, mette a disposizione dello sviluppatore un'utilità che consente di attivare un **monitor seriale** direttamente dalla shell MSYS2. Per usufruire di questa utilità è sufficiente lanciare il comando make con l'opzione monitor dalla cartella in cui è presente l'applicativo:

```
cd ~/esp/hello_world
make monitor
```

A questo punto la finestra della shell si trasformerà in un terminale seriale, mostrando la sequenza di caratteri ricevuti dall'applicazione (**Figura 14**).

Per uscire dal monitor è sufficiente premere la sequenza di tasti Ctrl - ], mentre Ctrl - T permette di richiamare l'help. Ogni altra sequenza di caratteri viene ignorata dal monitor. È inoltre possibile attivare il monitor contestualmente alla compilazione e all'upload tramite il comando:

```
make flash monitor
```

## AGGIORNAMENTO DELL'SDK

Il progetto ESP-IDF è in continua evoluzione e sottoposto a frequenti aggiornamenti, sia per correggere eventuali bachi, sia per aggiungere nuove funzionalità all'SDK.
È quindi preferibile tenere l'**ambiente costantemente aggiornato**, in modo da utilizzare sempre la versione più recente.
Quest'operazione è molto semplice, gestita direttamente dal tool Git al quale occorre fornire i seguenti comandi dalla shell MSYS2:

```
cd ~/esp/esp-idf
git pull
git submodule update --init --recursive
```

## CONCLUSIONI

Abbiamo visto in questo articolo come eseguire l'installazione dell'ambiente di sviluppo ESP-IDF, ovvero l'SDK ufficiale per il modulo ESP32 e quindi anche per la board ESPertino.
Informazioni dettagliate relative all'utilizzo delle librerie e delle API esportate dall'SDK sono disponibili a questo indirizzo, corredate di numerosi esempi applicativi.
In futuro torneremo ad occuparci di questo ambiente di sviluppo, proponendo nuove e interessanti applicazioni pratiche per la scheda ESPertino.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/sdk-per-espertino-guida-allinstallazione

# ESPERTINO E IFTTT: UN MAGGIORDOMO TUTTOFARE CON IL NOSTRO SMARTPHONE

di **Giovanni Di Maria**

*Vi piacerebbe accendere lo scaldabagno di casa con la semplice pressione di un'icona dello smartphone? Oppure, ancora, aprire e chiudere il cancello della propria villetta o spegnere le luci da qualsiasi posto vi troviate, sempre con l'ausilio del cellulare? Tutto questo è possibile grazie all'utilizzo di ESPertino, IFTTT e una Widget dello smart, creata ad hoc per lo scopo.*

## INTRODUZIONE

Ai giorni d'oggi, le applicazioni di domotica se ne contano a migliaia, di tutti i generi. Le possibilità di **comandare, a distanza, le utenze domestiche** sono, ormai, una realtà ben consolidata. Quella che proponiamo qui può considerarsi un'alternativa interessante focalizzata, soprattutto, all'utilizzo congiunto della scheda ESPertino e IFTTT. Il sistema utilizza anche lo smartphone per inoltrare i comandi, attraverso delle icone create appositamente per lo scopo.

## FINALITÀ DEL SISTEMA

Il progetto che stiamo andando a proporre può rivelarsi utile in mille occasioni. Le sue finalità sono generiche e ogni utente può adattarlo alle proprie esigenze. Il sistema è composto da una unità di comando, rappresentata dal proprio smartphone e da una remota ricevente, formata dalla nostra scheda ESPertino. Tra queste due dà il proprio contributo il servizio di IFTTT.

Come si evince dalla figura 1, l'utente ha a disposizione due icone sul telefonino che può utilizzare per attivare o disattivare un carico o un'utenza collegata alla centralina di ricezione. E' importante notare che **queste due icone non aprono una APP** sullo smartphone (come, ad esempio, un browser o un pannello di controllo) ma eseguono semplicemente un'Applet, in modo trasparente e invisibile. La loro creazione, infatti, si basa sulle Widgets di IFTTT.

## ALCUNI UTILIZZI PRATICI

Dal momento che il nostro maggiordomo elettronico remoto può **pilotare qualunque tipologia di carico**,

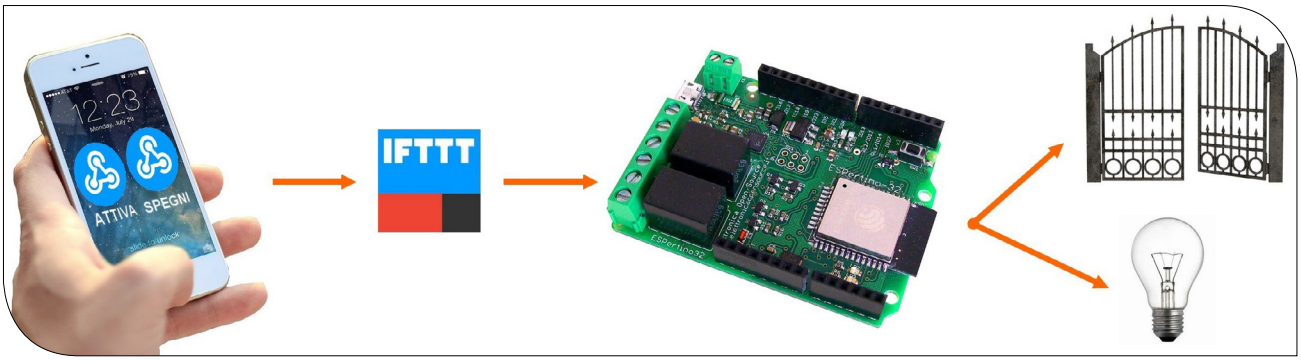



*Figura 1: lo schema di principio del sistema proposto*

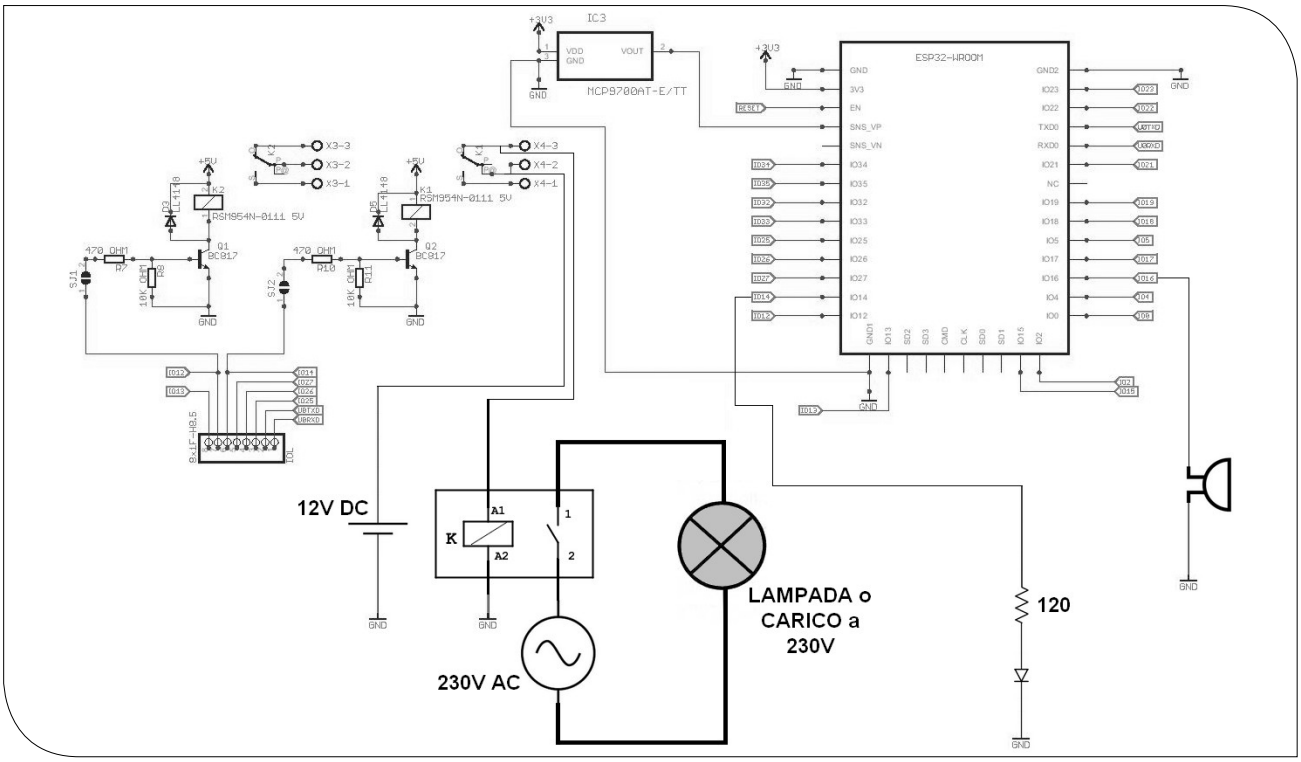



*Figura 2: lo schema elettrico del maggiordomo remoto*

i suoi utilizzi sono, praticamente, infiniti. Tuttavia elenchiamo solo alcune possibilità di applicazioni pratiche:

- accendere o spegnere lo scaldabagno a distanza;
- aprire il cancello della propria villetta;
- attivare o disattivare le luci del giardino o di casa quando si sta per rincasare;
- accendere la televisione, la radio o le luci a distanza per simulare la propria presenza in casa e far desistere i malintenzionati dal furto;
- attivare la saracinesca;
- innescare la pompa per il pescaggio dell'acqua, quando si è lontani;
- e mille altre idee.

## SCHEMA ELETTRICO

La figura 2 mostra il semplice schema elettrico. Esso è formato da:

- la scheda ESPertino;
- un diodo Led di segnalazione collegato alla porta GPIO 14, che è quella che fa capo anche al relé che si trova sulla stessa scheda;
- un buzzer di avviso di commutazione, collegato alla porta GPIO 16;
- un relé esterno, di potenza, per il pilotaggio di **carichi molto robusti**, funzionanti anche alla tensione alternata di 230V.

Il diodo Led è preceduto da una resistenza di limitazione, il cui valore è calcolato con **la legge di Ohm**. Il carico di potenza può essere costituito da qualunque attuatore: lampada a neon, a incandescenza, motore, lavatrice, forno, cancello, ecc. Il progettista deve solo dimensionare adeguatamente i dispositivi di commutazione.

## IL CABLAGGIO

In figura 3 è possibile osservare il cablaggio del sistema. Se si prevede un'installazione definitiva, si consiglia di allestire il circuito **all'interno di una scatola di plastica**, sotto la completa copertura del segnale WiFi. Le soluzioni per il carico sono molteplici. Se si prevede di comandare degli utilizzatori "leggeri" si può utilizzare solo il relé in dotazione con la scheda di ESPertino. Al limite è possibile implementare un sistema a **transistor o a Mosfet**. Viceversa occorre scegliere un relé molto robusto. La funzione del buzzer è quella di avvertire l'utente dell'avvenuta commutazione di stato del carico. Come vedremo più avanti, le durate dei suoi suoni prodotti sono differenti, a seconda che l'attuatore si attivi o meno. Anche il diodo Led ha esclusiva funzionalità di controllo. Non è indispensabile.

## LO SCRIPT IN PHP

E' necessario disporre di un **server Web** nel quale "de-

```
#include <WiFi.h>
//------------------WiFi---------------------
char* ssid          = "NETGEAR-gio22";
const char* password  = "miapassword";
WiFiClient client;       // Crea oggetto client
const char* host     = "www.elektrosoft.it";
String url;
char c;
String pacchetto;
//--------------Variabili-----------------------
int Rele        = 14;          // Porta del Rele 1 di bordo
int Buzzer      = 16;          // Porta del buzzer
int flag_stato   = 0;
void setup() {
   Serial.begin(9600);         // Comunicazione seriale a 9600 baud
   pinMode(Rele,OUTPUT);
   pinMode(Buzzer,OUTPUT);
   digitalWrite(Rele,LOW);       // Spegne il Rele
   digitalWrite(Buzzer,LOW);     // Spegne il Buzzer
   //----------Aspetta alcuni secondi per sincronizzare il WiFi----------
   Serial.println();
   Serial.println("Attesa per il WiFi...");
   WiFi.begin(ssid, password);
   delay(10000);
   Serial.println(WiFi.localIP());    // Visualizza l'indirizzo IP di ESPertino
   Serial.println("OK... pronto");
   //---------------------Fine WiFi-----------------
}
void loop() {
   //-------Prende il contenuto del file remoto maggiordomo.txt----------
   url = "/public/maggiordomo.txt";
   if(client.connect(host, 80)) {
      Serial.println("CONNESSO.....");
      client.print((String)"GET " + url + " HTTP/1.1\r\n" + "Host: " + String(host) + "\r\n" + "Connection: close\r\n\
r\n");
      //---------Aspetta fino a quando arrivano caratteri dalla pagina web---------
      while (!client.available());
      //----Costruisce una stringa con tutto il contenuto di cio' che arriva dal Web-------
      pacchetto="";
      while (client.available()) {
         c = client.read();
         pacchetto+=c;
      }
      client.stop();
      Serial.println(pacchetto);
      Serial.println(pacchetto.length());
      Serial.println("=============================================");
   }
   //-----------Se dentro la risposta del Server c'e' la frase 'CARICO ATTIVATO'...-------
   if(pacchetto.indexOf("CARICO ATTIVATO") >= 0 && flag_stato==0) {
      digitalWrite(Rele,HIGH);
      digitalWrite(Buzzer,HIGH);
      delay(500);
      digitalWrite(Buzzer,LOW);
      flag_stato=1;
   }
   //-----------Se dentro la risposta del Server c'e' la frase 'CARICO DISATTIVATO'...-------
   if(pacchetto.indexOf("CARICO DISATTIVATO") >= 0 && flag_stato==1) {
      digitalWrite(Rele,LOW);
      digitalWrite(Buzzer,HIGH);
      delay(200);
      digitalWrite(Buzzer,LOW);
      flag_stato=0;
   }
   delay(10000);
}
```

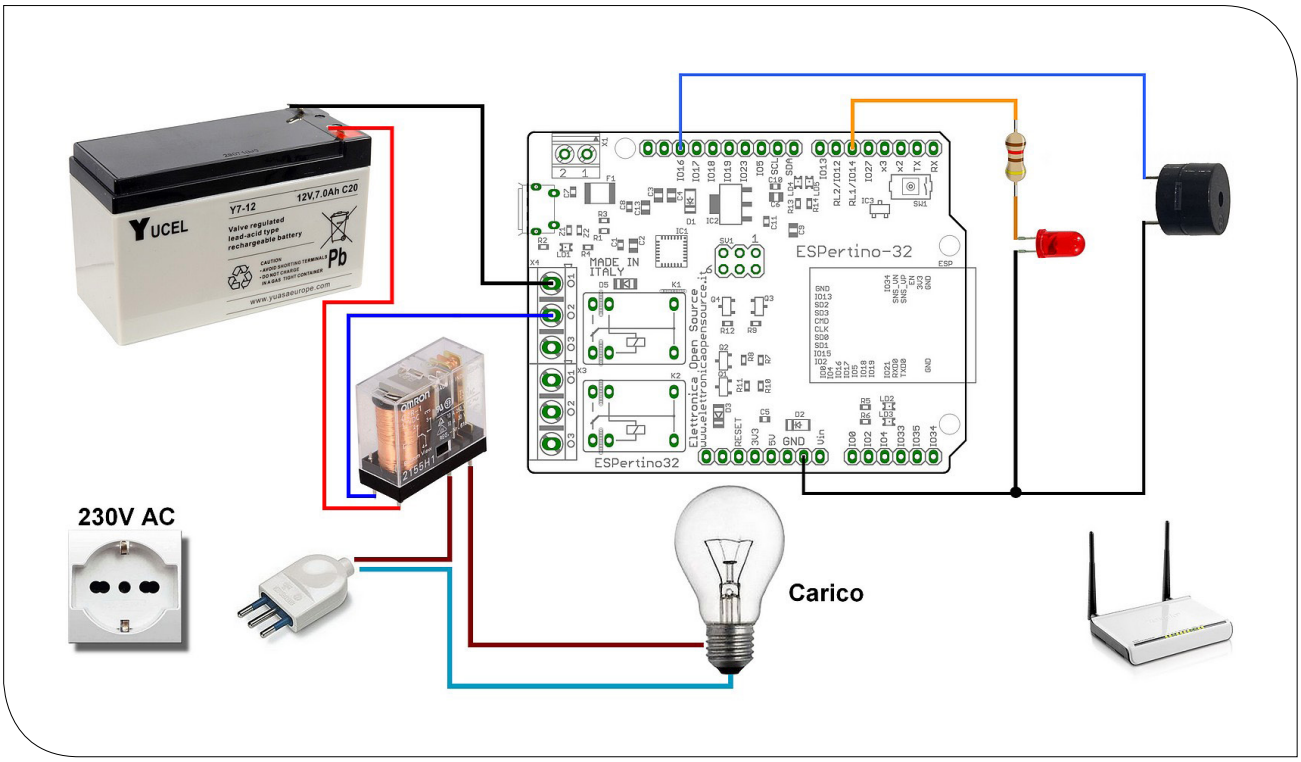



*Figura 3: il cablaggio del maggiordomo elettronico*

positare" un file di testo con lo stato logico dell'attuatore. Esso sarà modificato da uno script, in linguaggio PHP, da richiamarsi all'occorrenza attraverso una richiesta HTTP. Si deve creare uno script in PHP, a cui diamo il nome di "maggiordomo.php" che, quando invocato attraverso la chiamata di una URL, **memorizzi in un file** un particolare contenuto. L'indirizzo chiamato deve anche contenere un parametro GET che lo script stesso controlla e sceglie le informazioni da memorizzare nell'archivio testuale. Con un esempio chiariamo meglio il concetto. Se la URL chiamata è la seguente:

**http://www.elektrosoft.it/public/maggiordomo.php?stato=1**

lo script in PHP provvederà a memorizzare nel file "maggiordomo.txt", contenuto nella stessa cartella remota, la stringa "CARICO ATTIVATO".

Se, invece, la URL chiamata è la seguente:
**http://www.elektrosoft.it/public/maggiordomo.php?stato=0**

lo script in PHP provvederà a memorizzare nel file "maggiordomo.txt" la stringa "CARICO DISATTIVATO". Sarà poi lo sketch di ESPertino che, tramite una semplice **operazione di parsing**, prenderà le decisioni del caso. Di seguito lo script, molto semplice.

```
<?php
  //------------ maggiordomo.php --------
  echo "Esecuzione dello script PHP <br>";
  $stato=$_GET["stato"];
  if($stato=="1") {
    $myfile = fopen("maggiordomo.txt", "w");
    fwrite($myfile,"CARICO ATTIVATO");
    fclose($myfile);
    echo "ON <br>";
    echo $stato;
  }
  if($stato=="0") {
    $myfile = fopen("maggiordomo.txt", "w");
    fwrite($myfile,"CARICO DISATTIVATO");
    fclose($myfile);
    echo "OFF <br>";
    echo $stato;
  }
?>
```

Come si vede, dopo aver visualizzato la parola "Esecuzione dello script PHP", il programma preleva il para-

metro dalla URL memorizzandolo nella variabile "stato". Se tale parametro è uguale a 1, la stringa "CARICO ATTIVATO" sarà memorizzata nel file "maggiordomo.txt", aperto in modalità di scrittura. Viceversa la frase memorizzata sarà "CARICO DISATTIVATO". Il contenuto di tale file di testo sarà, poi, esaminato da ESPertino, come stringa di risposta dal server.
E' possibile provare subito lo script, inviando le due URL viste in precedenza. Si può anche controllare il contenuto del file di testo "maggiordomo.txt", inoltrando, su un browser, l'indirizzo:

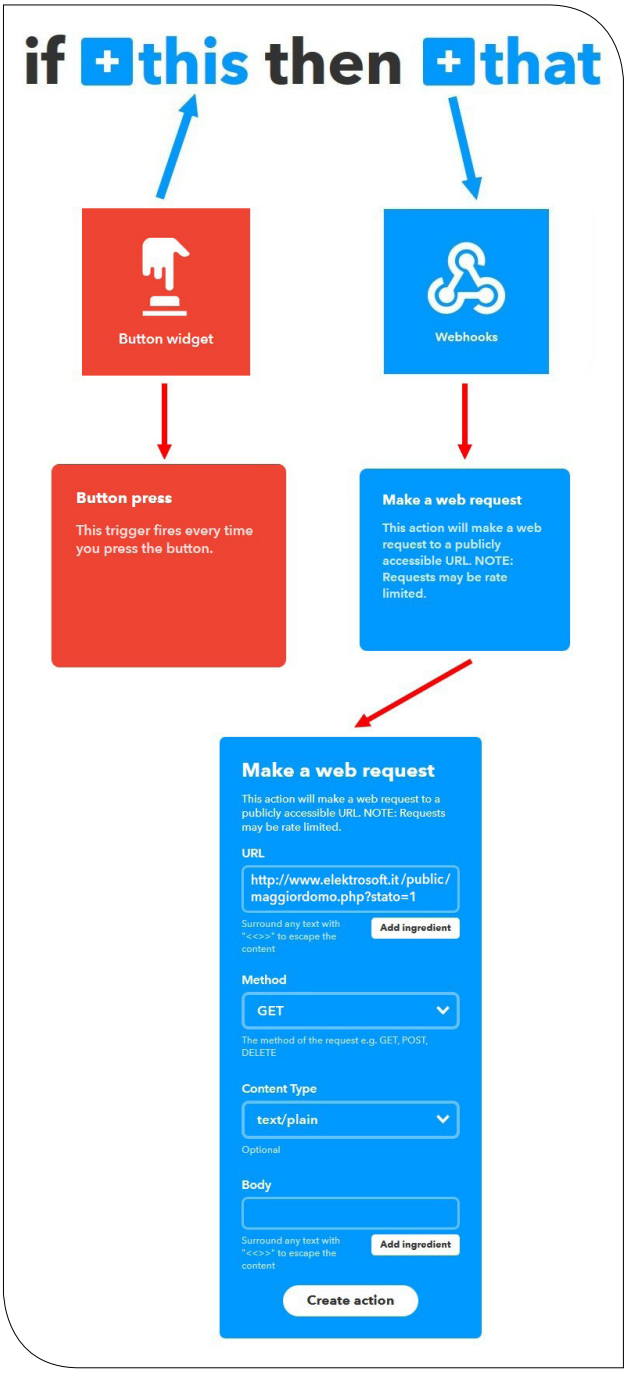



*Figura 4: la procedura per la creazione dell'applet*

**http://www.elektrosoft.it/public/maggiordomo.txt**

Si ricorda che l'utente deve utilizzare l'host di cui ha il permesso in scrittura.

## ANDIAMO A CREARE LO SKETCH DI ESPERTINO

Possiamo adesso dedicarci alla scrittura del programma per la scheda ESPertino. E' abbastanza semplice ma occo Il listato inizia con l'inclusione del file header WiFi.h, necessario per il collegamento di ESPertino al WiFi e la configurazione delle credenziali per l'accesso al router. Si passa, quindi, alla inizializzazione delle due variabili contenenti, rispettivamente, il numero della porta a cui è collegato il relé e quella dove è connesso il buzzer. Segue la funzione **setup()**, che esegue, una volta soltanto, la configurazione della linea seriale per i messaggi di controllo per il debug. Inoltre imposta la funzionalità della porta di I/O e, infine, effettua il collegamento effettivo al WiFi, con una pausa di attesa di almeno dieci secondi. Una volta che il programmatore ha testato il sistema, tutte le funzione "Serial.println()" potranno essere eliminate. Quindi, ha luogo la funzione **loop()**, che si ripete ogni dieci secondi. In essa il programma ricostruisce, a ogni "giro", una stringa contenente **l'intera risposta inviata dal server** a ogni richiesta HTTP, dopo che il software interroga il file di testo "maggiordomo.txt".
Se all'interno del pacchetto anzi, più precisamente, in coda è presente la stringa "CARICO ATTIVATO", si attiva il relé (assieme al relativo diodo Led sulla porta 14) e il buzzer emette un suono per mezzo secondo.
Se, nell'altro caso, all'interno del pacchetto è presente la stringa "CARICO DISATTIVATO", il relé e il diodo Led si spengono e il buzzer emette un breve suono di 200 millisecondi. La presenza del "flag_stato" consente di ripetere le porzioni condizionali solo una volta.

## L'APPLET DI IFTTT

Andiamo a creare, adesso, le due applets su IFTTT. In pratica esse sono identiche tra loro, cambia solo il parametro della URL passata che, rispettivamente, attiva o disattiva il carico. Come condizione **"if this"**, stavolta andiamo a selezionare il servizio **Button widget**. La figura 4 mostra l'intera procedura da seguire. Questo servizio è innescato ogni volta che sul proprio smartphone viene pressata un'apposita icona, creata attraverso il widget di IFTTT. Nel prossimo paragrafo esamineremo tale procedura. Come azione **"that"** scegliamo il servizio **Webhooks**, già esaminato in altri nostri precedenti articoli. Esso effettuerà delle richieste web HTTP verso

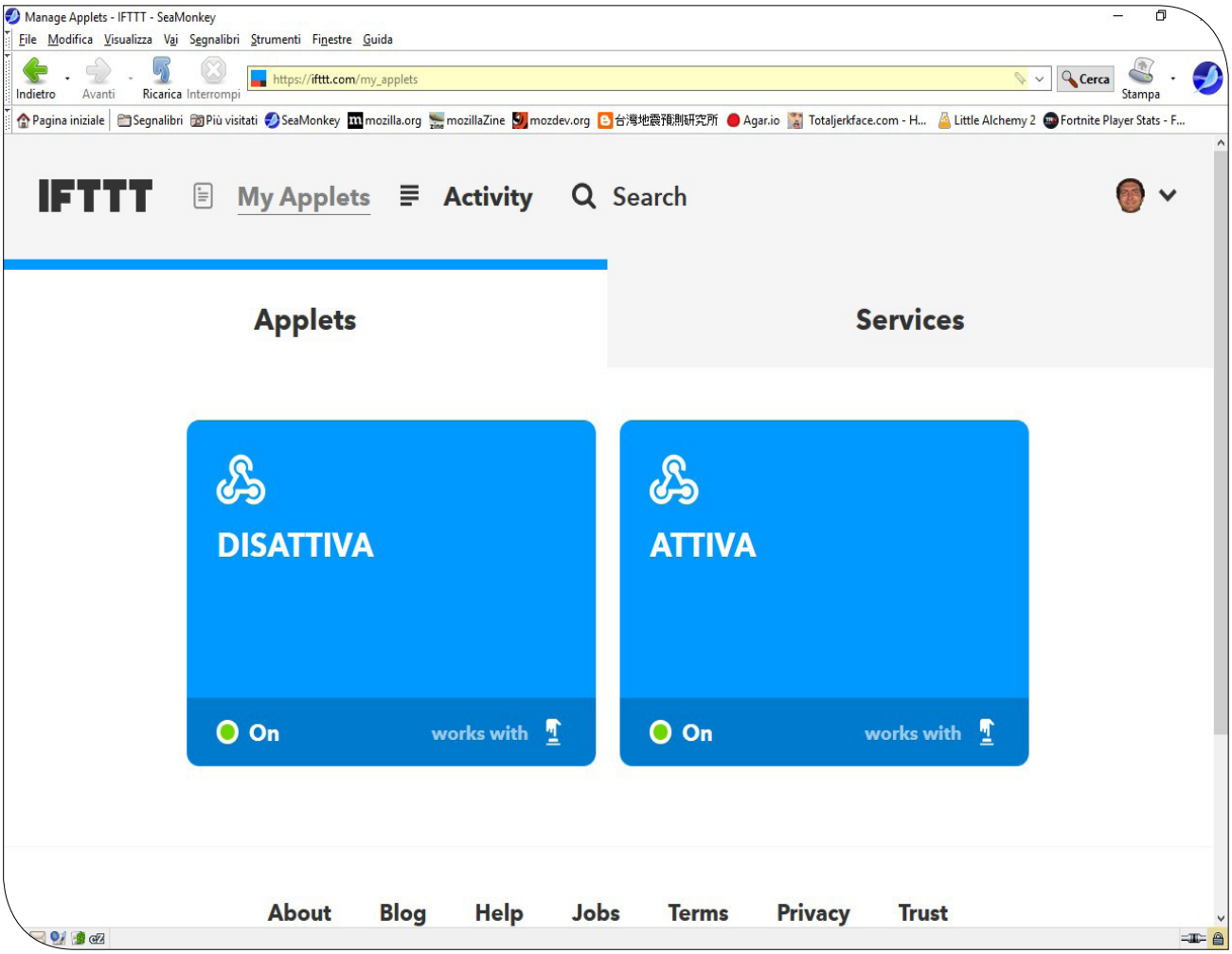



*Figura 5: le due applets create*

gli indirizzi prima esaminati. rre focalizzare alcuni punti critici. Occorre creare due applet con la stessa filosofia di funzionamento. Alla prima, alla quale si attribuirà il nome "ATTIVA" si deve passare la seguente URL:

**http://www.elektrosoft.it/public/maggiordomo.php?stato=1**

Alla seconda, alla quale si attribuirà il nome "DISATTIVA" si deve passare quest'altra URL:

**http://www.elektrosoft.it/public/maggiordomo.php?stato=0**

Per la prossima fase teniamo a mente tali nomi. In figura 5 è possibile osservare entrambe le applets create.

## LA CREAZIONE DELLE ICONE SULLO SMARTPHONE

Accediamo, adesso, alla scheda delle Widgets del nostro smartphone. Ricordiamo che per l'implementazione e il funzionamento della procedura **deve essere installata l'applicazione IFTTT** con le proprie credenziali. In dettaglio occorre seguire le seguenti fasi:

1. si selezioni l'incona "Applicazioni";
2. si faccia "tap" sulla scheda "Widget";
3. si scorrano le pagine fino a trovare le widget di IFTTT. Allo stato attuale esse sono due: la "small" e la "large", come si può evincere dalla figura 6;
4. si selezioni la widget preferita tenendola premuta per qualche istante;
5. la si trascini nell'area desiderata;
6. il sistema, adesso, chiede di associarla a una delle due applets create in precedenza, come visibile in figura 7. Si scelga, come prima volta, l'applet "ATTIVA";
7. si ripeta la procedura per l'altra applet.

FInalmente, alla fine lo smartphone conterrà due nuove icone, come mostrato in figura 8. La prima serve per

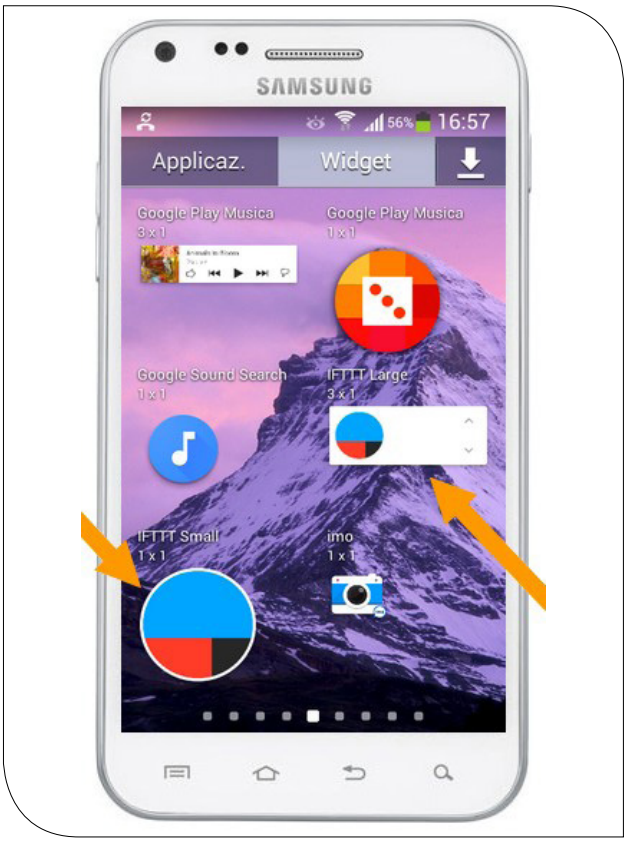



Figura 6: le due widgets di IFTTT

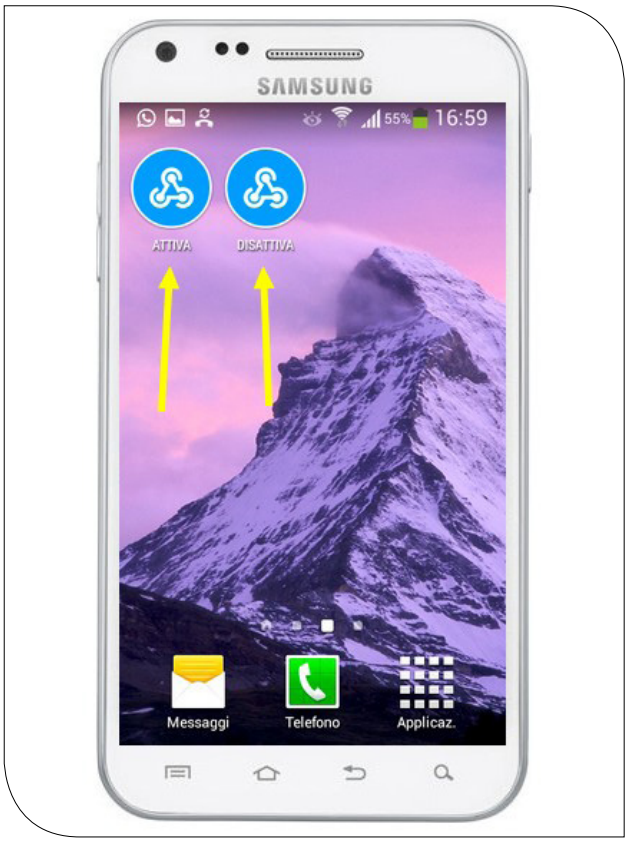



Figura 8: le due icone create sullo smartphone

Figura 7: l'associazione delle widgets con le applets

attivare il carico remoto, la seconda per spegnerlo. Entrambe hanno il contrassegno del servizio Webhooks. E' interessante notare che la selezione di una delle due icone **non apre alcuna APP**. Al contrario essa esegue il servizio in modo trasparente.

## COLLAUDO

Finalmente siamo pronti per il tanto atteso collaudo. Si colleghi, a casa, ESPertino a una fonte di energia. Per il pilotaggio del relé e dei carichi si prestino le dovute precauzioni. Ci vuole veramente poco per mandare tutto in fumo, compresa la nostra amata scheda. Ci si assicuri, altresì, che **l'embedded sia coperto da un sufficiente segnale WiFi**, proveniente da un router collegato a Internet. Si accenda, dunque, il sistema. Dopo dieci secondi esso è pronto. I carichi e gli attuatori collegati dovrebbero risultare disattivati. Si passi, adesso, allo smartphone. Si faccia "tap" sull'icona "ATTIVA". Dopo una decina di secondi il carico dovrebbe attivarsi e tale procedura dovrebbe essere testimoniata anche da un beep sonoro. Si inoltri, quindi, un "tap" sull'icona "DISATTIVA". Dopo una decina di secondi l'attuatore dovrebbe spegnersi. Si è scelta questa tempistica un po' lunga proprio **per non stressare il server**.

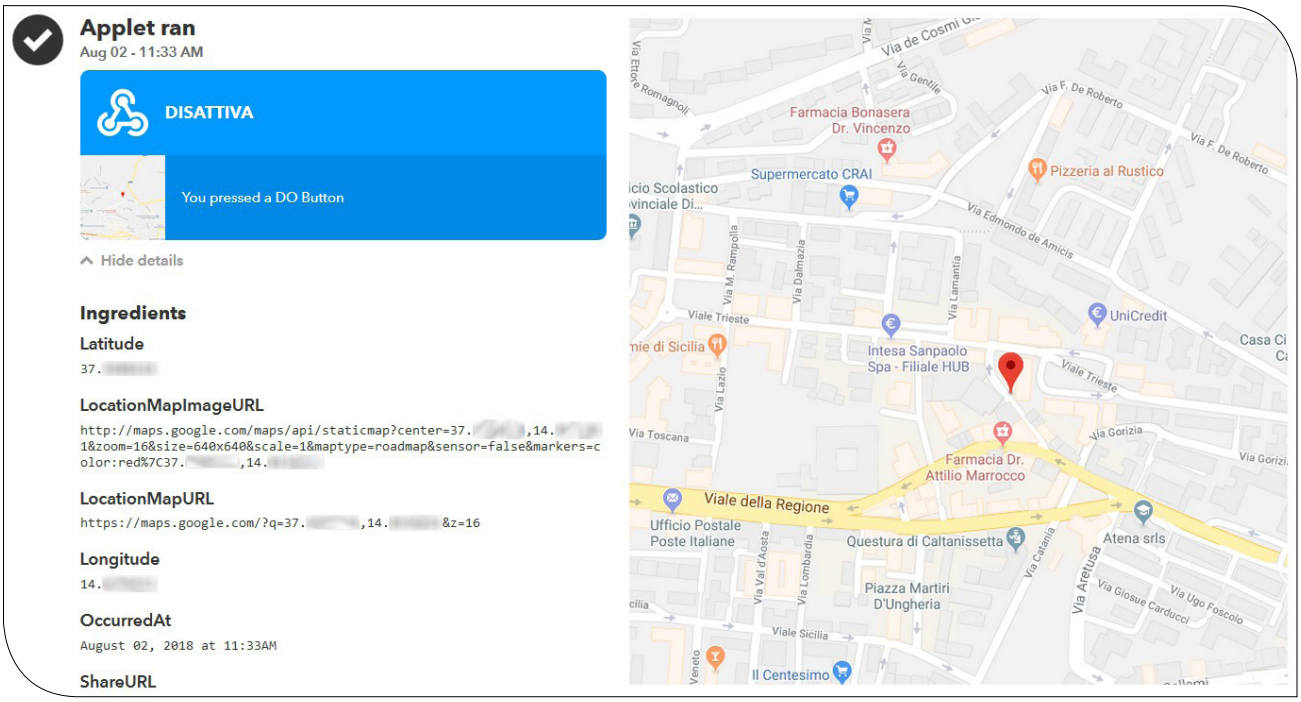



*Figura 9: IFTTT registra anche le coordinate geografiche degli eventi*

Lo sketch, infatti, esegue un continuo polling sulla rete. Si noti che dopo aver toccato le icone sullo schermo del cellulare esse modificano, per un attimo, la propria grafica, mostrando delle frecce rotanti.
E' interessante notare che lo smartphone può eseguire il comando dei carichi da qualsiasi posto sul pianeta.

## IFTTT CONTROLLA TUTTO

Accedendo all'elenco delle attività svolte dai propri servizi di IFTTT (Activity) è possibile scoprire come esso, effettivamente, controlli tutte le nostre mosse.
Ad esempio cliccando, al PC, i dettagli di un'azione svolta dalle applets (Show details), è possibile osservare un **elenco dettagliato degli ingredienti** coinvolti nell'operazione.
Esso riporta, tra gli altri, la latitudine e la longitudine della locazione in cui si trovava lo smartphone al momento dell'attivazione o della disattivazione del carico.
Inoltre, il sistema propone anche l'indirizzo di **Google Maps**, che può essere copiato e incollato sul browser per scoprire, visivamente, dove ci si trovava al momento della pressione delle icone, come mostrato in figura 9.

**>>>Leggi anche Realizziamo applicazioni IoT con il Raspberry Pi 3: primo approccio**

## CONCLUSIONI

Non c'è bisogno di puntualizzare l'importanza del progetto che abbiamo appena proposto in questo articolo.
La sua utilità può rivelarsi fondamentale in qualsiasi campo di applicazione: ludica, commerciale e domotica.
La parte da leone, indubbiamente, la fa la scheda ESPertino in quanto, con il relé e il WiFi a bordo, non esige alcun componente esterno o altra scheda aggiuntiva. Un montaggio minimale, insomma, che toglie al progettista tanto lavoro dal punto di vista hardware.
La finalità del progetto, ovviamente, **può essere modificata a piacere**, secondo le proprie esigenze. Possono, ad esempio, essere aggiunte altre icone di comando oppure, ancora, diversificati i carichi presenti sulla scheda.
Insomma, oggi è ancor più vero il fatto che, con la scheda ESPertino, un collegamento a Internet, uno smartphone e i servizi IFTTT, si può controllare il mondo con grande facilità.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/espertino-e-ifttt-un-maggiordomo-tuttofare-con-il-nostro-smartphone

# ESPERTINO E IFTTT: PULSANTONE DI EMERGENZA PER ANZIANI

di **Giovanni Di Maria**

*In questo articolo proponiamo la realizzazione di un semplice pulsante che potrebbe risultare utile in molte situazioni se non, addirittura, salvare la vita. Si tratta di un comando remoto, a disposizione di persone parzialmente autosufficienti che, se azionato, avvia molteplici chiamate di emergenza verso diversi canali di comunicazione. Il dispositivo presentato costituisce una versione base del progetto. I lettori possono, naturalmente, espanderlo e integrarlo ad altre applicazioni, secondo le proprie necessità ed esigenze.*

## LA SOLITUDINE È NOIOSA E, SPESSO, PERICOLOSA

Gli anziani, molto spesso, vengono lasciati a casa da soli. Specialmente nei periodi estivi tali episodi sono estremamente ricorrenti. Sino a quando non succede nulla e la situazione procede nella normalità, si è tutti felici e contenti. Ma, a volte, potrebbe capitare il peggio: un malore, un principio d'incendio, una caduta dalla sedia, un dolore improvviso sono tutti episodi che **devono essere notificati immediatamente alle persone più care**. Più l'intervento è repentino e immediato e maggiori sono le probabilità di risolvere il problema nel migliore dei modi. Non si tratta di fatti che accadono solo agli altri e che si sentono esclusivamente nei telegiornali. Essi possono realmente capitare a tutti. Un sistema di emergenza per anziani costituisce, oggi, un dispositivo molto semplice da realizzare. La disponibilità d'Internet e di altri sistemi di comunicazione (cellulari, tablet, avvisi di chiamata e altri) consentono di allestire ottimi progetti di notifica.

## IL PROGETTO

Quello presentato in questo articolo utilizza due pilastri per la realizzazione di dispositivi IoT:

- la scheda di sviluppo ESPertino;
- i servizi offerti da IFTTT.

E' necessario disporre, altresì, di un collegamento a Internet tramite Router WiFi, che ESPertino sa gestire egregiamente, tramite il suo ESP32. Alcuni componenti elettronici esterni minimali completano la costruzione.

Prima d'iniziare la realizzazione occorre avere, ben chiare, le procedure di emergenza che il circuito deve intraprendere, in caso di richiesta d'aiuto. Attraverso i servizi offerti da IFTTT si è pensato di utilizzare i seguenti:

- un invio immediato di un **messaggio su Mes-**

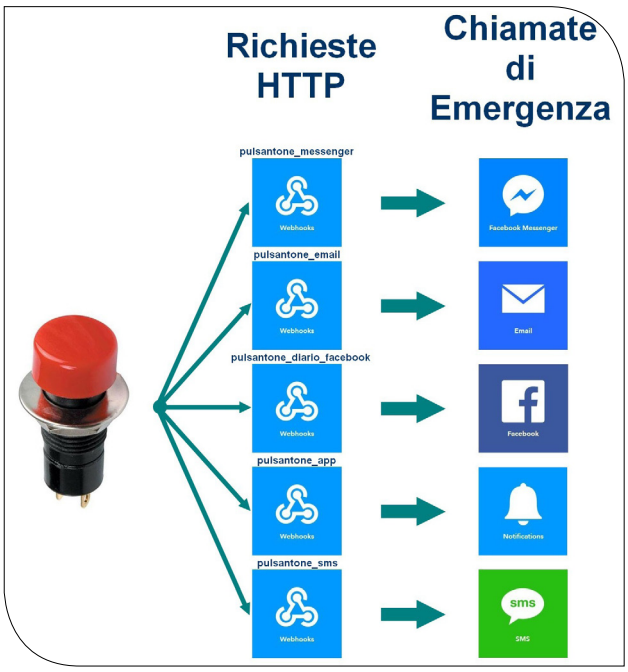



*Figura 1: il diagramma concettuale del progetto del pulsante di emergenza*

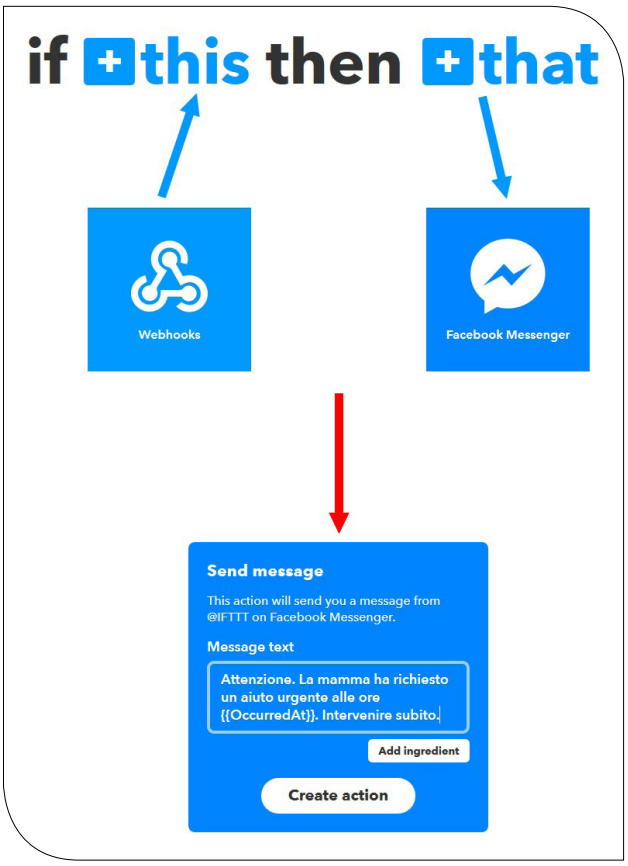



*Figura 2: l'applet che prevede l'invio di un messaggio su Messenger in caso di richiesta d'aiuto*

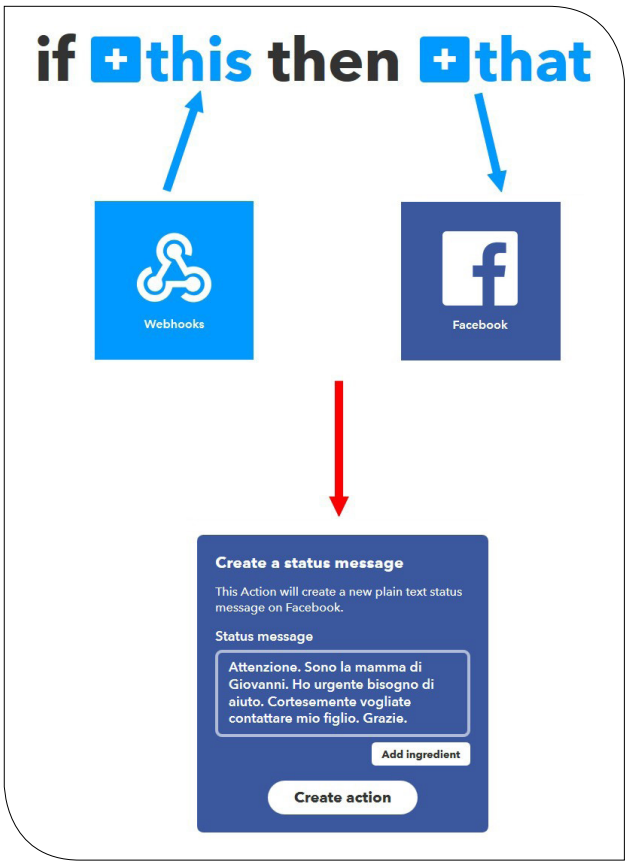



*Figura 4: l'applet che prevede l'invio di un messaggio sul diario di Facebook in caso di richiesta d'aiuto*

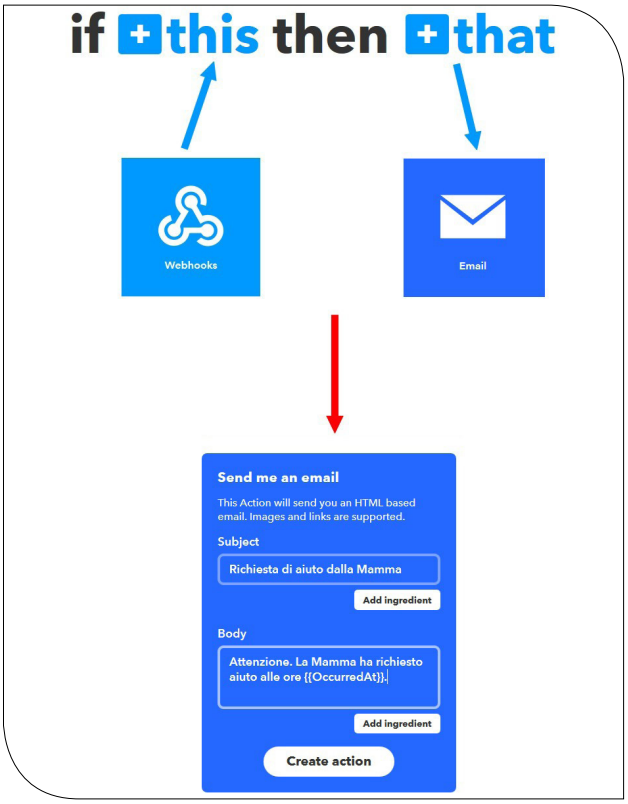



*Figura 3: l'applet che prevede l'invio di un messaggio in email in caso di richiesta d'aiuto*

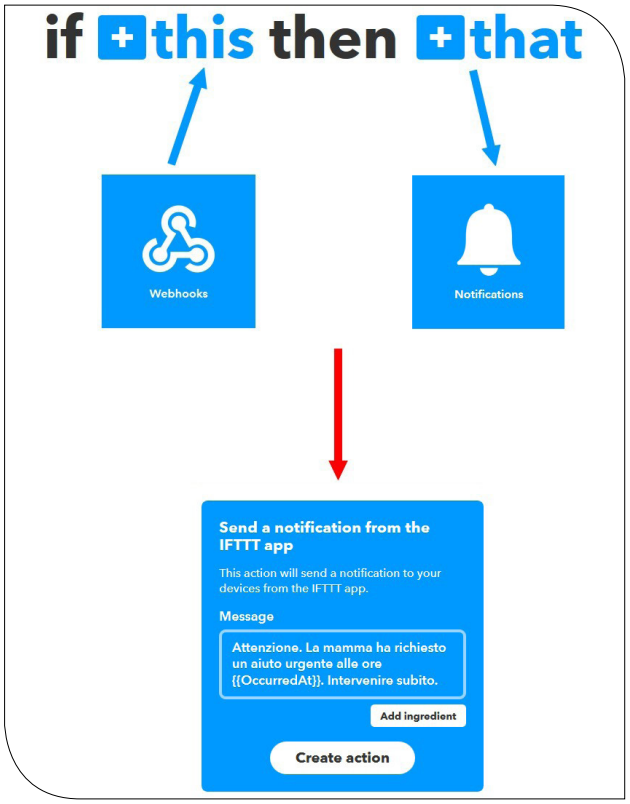



*Figura 5: l'applet che prevede l'invio di un messaggio sull'APP di IFTTT in caso di richiesta d'aiuto*

**senger**. Esso assicura un rapidissimo raggiungimento del destinatario con la garanzia di una rapida lettura;
- un invio di una **email** all'indirizzo del destinatario;
- un invio di un **messaggio sul diario di Facebook**, in modo che tutti gli amici del destinatario possano leggerlo ed, eventualmente, cercare di contattare il parente di chi richiede il soccorso;
- una **notifica** sul proprio smartphone all'applicazione IFTTT (che deve essere installata sul cellulare);
- un **messaggio SMS** su un telefono cellulare di un parente stretto.

Come si può notare, in caso di emergenza, le notifiche e le richieste d'aiuto sono molteplici e differenziati ed è davvero difficile che i messaggi di domanda d'intervento siano ignorati. Il non ascolto e la non considerazione di essi sono davvero improbabili.
La figura 1 illustra lo schema logico che organizza il progetto. Un pulsante, di dimensioni importanti, è collegato alla scheda ESPertino. La sua attivazione va a produrre alcune richieste di tipo HTTP (grazie ai trigger di IFTTT) che, a loro volta, **innescano tutte le azioni programmate**.

## ANDIAMO A LAVORARE CON IFTTT

Bene, siamo pronti per creare i trigger e le azioni con IFTTT. L'innesco generale è costituito da alcune richieste di tipo HTTP, generate dalla scheda ESPertino con la pressione del pulsantone di emergenza.

## APPLET CHE INVIA UN MESSAGGIO SUL MESSENGER

Iniziamo a preparare la prima applet adibita all'inoltro di un messaggio privato su Messenger. Esso, naturalmente, sarà ricevuto anche sul PC attraverso Facebook. La condizione scatenante sarà, come detto prima, la spedizione di una richiesta HTTP da parte di ESPertino. Essa è "programmata" grazie all'uso del **servizio Webhooks**. Decidiamo, inoltre, di dare il nome "pulsantone_messenger" all'evento. Esso servirà per la sua successiva invocazione. Un messaggio ad hoc completa la creazione dell'applet. Nel nostro caso esso è:

**"Attenzione. La mamma ha richiesto un aiuto urgente alle ore {{OccurredAt}}. Intervenire subito".**

L'ingrediente {{OccurredAt}} funziona proprio come una variabile, in quanto reperisce l'ora corrente le la include nel testo da inviare. E' possibile corredarlo di altre informazioni, o **ingredienti**, come l'ora dell'invio. La figura 2 mostra il flusso del trigger e dell'azione.
La prima volta che si effettua l'attivazione del Messenger è richiesta una conferma, via email, della paternità dell'account.

## APPLET CHE INVIA UN MESSAGGIO VIA EMAIL

La seconda applet che serve è quella preposta all'invio di un messaggio via email, in caso d'inoltro della richiesta di aiuto. Anche in questo caso, la condizione d'innesco sarà l'invio di una richiesta HTTP da parte di ESPertino. Essa è realizzata sempre con il **servizio Webhooks**. Il nome dell'evento da noi attribuito è "pulsantone_messenger". Come si vede in figura 3 l'applet è completata con l'inserimento dell'oggetto e del corpo della email. In quest'ultimo caso si possono inserire delle informazioni variabili, come l'ora dell'evento, grazie all'adozione degli ingredienti.

## APPLET CHE INVIA UN MESSAGGIO SUL DIARIO DI FACEBOOK

La terza applet che iniziamo a preparare è "più pubblica" e visibile delle precedenti. Il messaggio di "status" inviato **sarà letto, infatti, da tutti gli amici del famoso social network**. Il nome dell'evento da noi attribuito è "pulsantone_diario_facebook". La figura 4 mostra le fasi di creazione dell'applet.

## APPLET CHE INVIA UNA NOTIFICA SULLO SMARTPHONE CON L'APP DI IFTTT

La quarta applet che serve è quella che prevede una notifica all'APP di IFTTT presente sul proprio smartphone. Allo scopo, dunque, essa deve essere installata con le proprie credenziali. La condizione d'innesco sarà, come per le precedenti, l'invio di una richiesta HTTP da parte di ESPertino. Il nome dell'evento da noi attribuito è "pulsantone_app". La figura 5 mostra tutte le caratteristiche operative. L'azione innescata è di tipo "Notifications". Anche in questo caso **il messaggio può essere personalizzato e parametrizzato** grazie all'uso degli ingredienti.

## APPLET CHE INVIA UN MESSAGGIO SMS SUL TELEFONINO

Probabilmente la quinta applet è la più interessante di tutte. Essa è quella preposta all'invio di un messaggio SMS al telefonino in caso d'inoltro di una richiesta di

```
1   #include <WiFi.h>
2   //------------------WiFi---------------------
3   char* ssid          = "NETGEAR-gio22";
4   const char* password  = "XXXXXXXXXXXXXXX";
5   WiFiClient client;       // Crea un oggetto client
6   const char* host     = "maker.ifttt.com";
7   String url;
8   //--------------Variabili-----------------------
9   int Pulsantone  = 34;          // Numero della porta del pulsantone
10   int Led        = 16;          // Numero della porta del diodo Led esterno
11   void setup() {
12       Serial.begin(9600);         // Comunicazione seriale a 9600 baud
13       pinMode(Pulsantone,INPUT);   // Definisce in ingresso il Pin per il pulsante
14       pinMode(Led,OUTPUT);        // Definisce in in uscita il Pin per il diodo Led di segnalazione
15       digitalWrite(Led,LOW);       // Spegne il diodo Led
16       //----------Aspetta alcuni secondi per sincronizzare il WiFi----------
17       Serial.println();
18       Serial.println("Attesa per il WiFi...");
19       WiFi.begin(ssid, password);
20       delay(10000);
21       Serial.println(WiFi.localIP());    // Visualizza l'indirizzo IP di ESPertino
22       Serial.println("OK... pronto");
23       //---------------------Fine WiFi-----------------
24   }
25   void loop() {
26       if(digitalRead(Pulsantone)==HIGH) {        // Se il pulsantone viene premuto, anche per un istante...
27           digitalWrite(Led,HIGH);               // Si accende il diodo Led di Emergenza
28           Serial.println("Richiesta di Emergenza attivata..... inizio dell'invio delle richieste di aiuto...");
29           delay(3000);
30           //------------------------------- Messaggio di aiuto inviato su Messenger ------------
31           url = "/trigger/pulsantone_messenger/with/key/XXXXXXXXXXXXXXXXXXXX";
32           client.connect(host, 80);
33           client.print((String)"GET " + url + " HTTP/1.1\r\n" + "Host: " + String(host) + "\r\n" + "Connection: close\r\n\
r\n");
34           Serial.println("Messaggio inviato su MESSENGER.");
35           delay(3000);
36           //------------------------------- Messaggio di aiuto inviato sulla Email ------------
37           url = "/trigger/pulsantone_email/with/key/XXXXXXXXXXXXXXXXXXXX";
38           client.connect(host, 80);
39           client.print((String)"GET " + url + " HTTP/1.1\r\n" + "Host: " + String(host) + "\r\n" + "Connection: close\r\n\
r\n");
40           Serial.println("Messaggio inviato sulla Email.");
41           delay(3000);
42           //------------------------------- Messaggio di aiuto inviato sul diario di Facebook ------------
43           url = "/trigger/pulsantone_diario_facebook/with/key/XXXXXXXXXXXXXXXXXXXX";
44           client.connect(host, 80);
45           client.print((String)"GET " + url + " HTTP/1.1\r\n" + "Host: " + String(host) + "\r\n" + "Connection: close\r\n\
r\n");
46           Serial.println("Messaggio inviato sul diario di Facebook");
47           delay(3000);
48           //------------------------------- Messaggio di aiuto inviato sulla APP di IFTTT ------------
49           url = "/trigger/pulsantone_app/with/key/XXXXXXXXXXXXXXXXXXXX";
50           client.connect(host, 80);
51           client.print((String)"GET " + url + " HTTP/1.1\r\n" + "Host: " + String(host) + "\r\n" + "Connection: close\r\n\
r\n");
52           Serial.println("Messaggio inviato sull'APP di IFTTT");
53           delay(3000);
54           //------------------------------- Messaggio di aiuto inviato su SMS ------------
55           url = "/trigger/pulsantone_sms/with/key/XXXXXXXXXXXXXXXXXXXX";
56           client.connect(host, 80);
57           client.print((String)"GET " + url + " HTTP/1.1\r\n" + "Host: " + String(host) + "\r\n" + "Connection: close\r\n\
r\n");
58           Serial.println("Messaggio inviato via SMS sul telefonino");
59           //-------------------------------------------------------------------------------
60           delay(3000);
61           Serial.println("Un'altra richiesta puo' essere effettuata tra 60 secondi...");
62           delay(60000);
63           digitalWrite(Led,LOW);       // Spegne il diodo Led
64           Serial.println("OK... pronto");
65       }
66       delay(100);
67   }
```

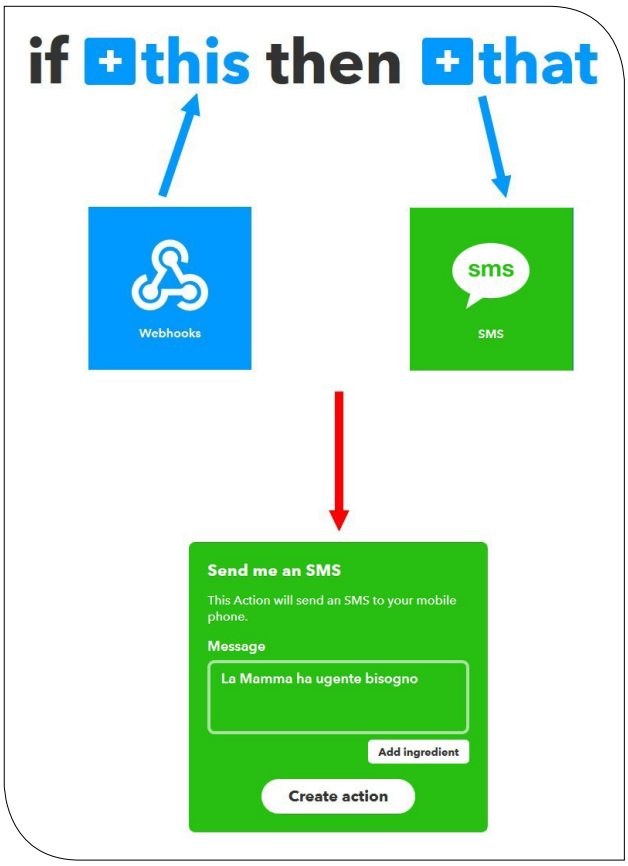



*Figura 6: l'applet che prevede l'invio di un messaggio SMS in caso di richiesta d'aiuto*

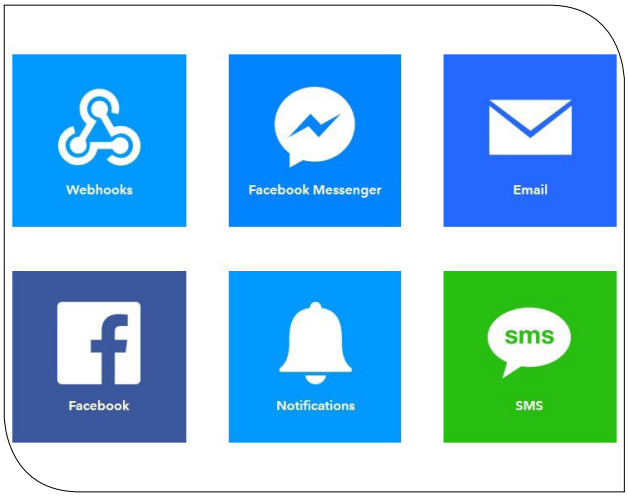



*Figura 7: i servizi utilizzati di IFTTT per la realizzazione del pulsante di emergenza*

aiuto. Al primo utilizzo occorre sincerarsi che il numero telefonico composto sia coperto dal servizio. La prima volta arriva anche un pin per confermare tale collegamento. Il nome affibbiato al trigger è "pulsantone_sms". E' consigliabile **non eccedere nel numero dei caratteri** del messaggio SMS (vedi figura 6) per non vedersi "frazionato" l'avviso in più invii. Occorre anche prestare attenzione al **numero massimo di SMS a disposizione** al mese, solitamente solo dieci. Un messaggio in email, da parte di IFTTT, ricorda quando gli SMS stanno per finire.

## I SERVIZI UTILIZZATI SU IFTTT E LE VARIE APPLETS CREATE

Riepilogando il lavoro preparativo svolto in precedenza, la figura 7 mostra tutti i servizi utilizzati su IFTTT. Si tratta di un sistema informativo molto potente che, in un solo movimento, coinvolge tanti canali di comunicazione.

Essi, nell'ordine d'inoltro previsto, sono i seguenti:

- **Webhooks**, che intercetta le chiamate HTTP;
- **Facebook Messenger**, che si dedica all'inoltro di un messaggio su Messenger;
- **Email**, che è preposto all'invio di una email;
- **Facebook**, che sovrintende alla scrittura di un messaggio di stato sul diario di Facebook;
- **Notifications**, che provvede all'invio di una notifica tramite APP IFTTT;
- **SMS**, che manda un SMS al numero di telefonino programmato.

La figura 8, invece, mostra le cinque applets create. Ognuno di esse espleta un preciso compito automatizzato. Come si vede dall'indicatore verde (ON) dei vari componenti, esse sono costantemente "in ascolto" del verificarsi dell'evento scatenante. All'occorrenza un'applet può essere disattivata temporaneamente oppure eliminata definitivamente.

## SCHEMA ELETTRICO

Come si vede dalla figura 9, lo schema elettrico risulta estremamente semplice e minimale. E' costituito, ovviamente, dalla scheda ESPertino, da un **sensore** e da un **attuatore**. Il sensore è rappresentato dal pulsantone che fa capo alla porta digitale d'ingresso GPIO 34. Una resistenza di pull-down assicura un potenziale logico basso in caso che il pulsante non venga premuto. In caso contrario, la tensione di 3.3V fluirà alla porta, commutando in ALTO il suo stato logico. Il diodo Led ha lo scopo esclusivo di mostrare, con la sua accensione, lo stato di chiamata in corso. In condizioni normali esso rimane spento. La resistenza di limitazione assicura che lo stesso semiconduttore non si danneggi, per l'eccessiva corrente.

In figura 10 è possibile osservare il piano di cablaggio del progetto. Come si può notare, si tratta solo di utilizzare due resistori, un pulsante e un diodo Led.

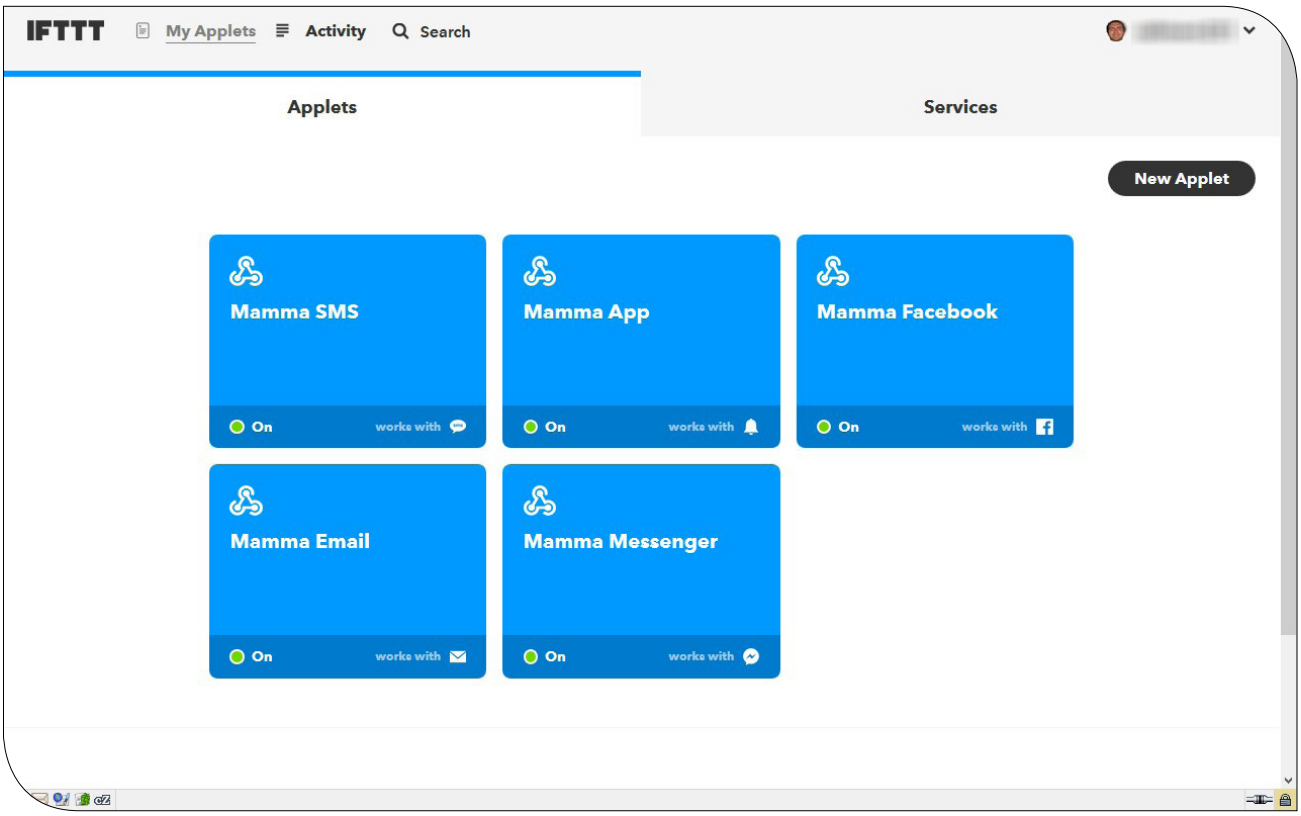



*Figura 8: le cinque applets create per il pulsante di emergenza*

## IL PULSANTONE DA UTILIZZARE

Sarebbe del tutto inutile utilizzare un piccolo pulsantino, difficile da scovare e da rintracciare per tutti. E' più opportuno usare un grande pulsante, dal diametro di almeno dieci centimetri, come quello visualizzato in figura 11. Le idee, in tal senso, possono essere le più disparate. Si potrebbe anche costruirlo con materiali di recupero, in modo che la sua forma e le dimensioni ricoprano perfettamente le esigenze del caso. A ogni modo il pulsante utilizzato, dal punto di vista elettrico, deve essere del tipo **normalmente aperto**.

## LO SKETCH

Adesso proponiamo e analizziamo il firmware da caricare, tramite l'Arduino IDE, sulla scheda ESPertino. In fase di digitazione non si devono includere i numeri di linea progressivi, che hanno solo la funzione di riferimento per i commenti successivi. Ecco, qui sotto, lo sketch completo.

Commentiamo, adesso, le parti salienti del listato, riferendoci ai numeri di linea:

- la riga 1 predispone l'inclusione del file header WiFi.h, necessario per il collegamento al WiFi;
- dalla riga 2 alla riga 7 è predisposta la configurazione delle credenziali del router;
- la riga 6 contiene una sequenza di caratteri con il nome dell'host del servizio di IFTTT. Non si deve assolutamente modificare;
- le riga 9 e 10 inizializzano le variabili contenenti il numero dei pin, rispettivamente, del pulsante (input) e del diodo Led (output);
- la funzione **setup()**, contenuta dalla riga 11 alla riga 24 esegue, una volta soltanto, la configurazione della linea seriale per i messaggi di controllo. Inoltre imposta la funzionalità delle due porte di I/O e, infine, effettua il collegamento al WiFi, con la canonica pausa di attesa di almeno dieci secondi;
- dalla riga 25 alla riga 67 ha luogo la funzione **loop()**, che si ripete ogni decimo di secondo per avere la certezza che la pressione del tasto (anche di breve durata) venga rilevata con sufficiente sicurezza. Pause più lunghe non garantirebbero l'evento;
- la riga 26 controlla se il pulsante è premuto, nel qual caso tutto il corpo della condizione viene eseguito, riga per riga;
- la riga 27 illumina il diodo Led, che attesta lo stato di emergenza in corso;
- dalla riga 30 alla riga 35 viene gestito il blocco

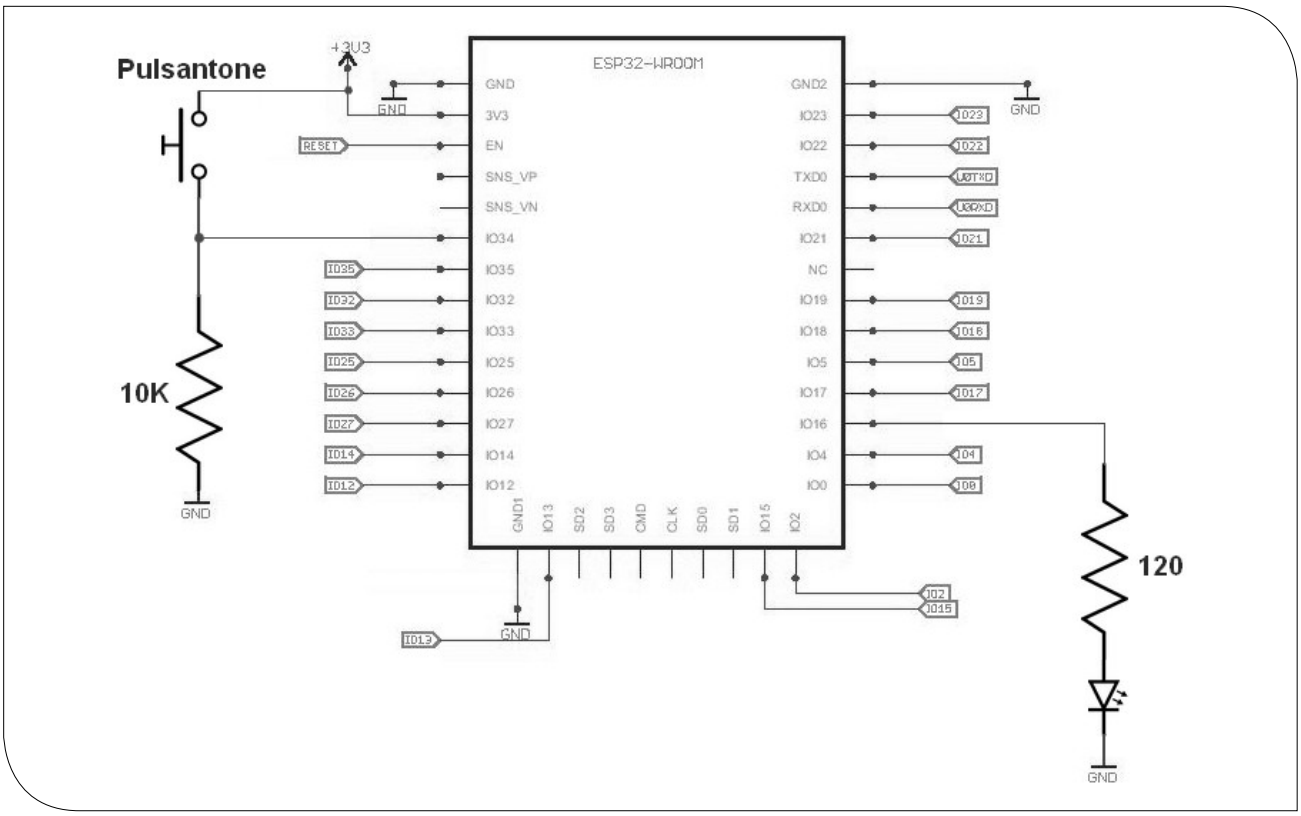



*Figura 9: il semplice schema elettrico del prototipo del pulsantone*

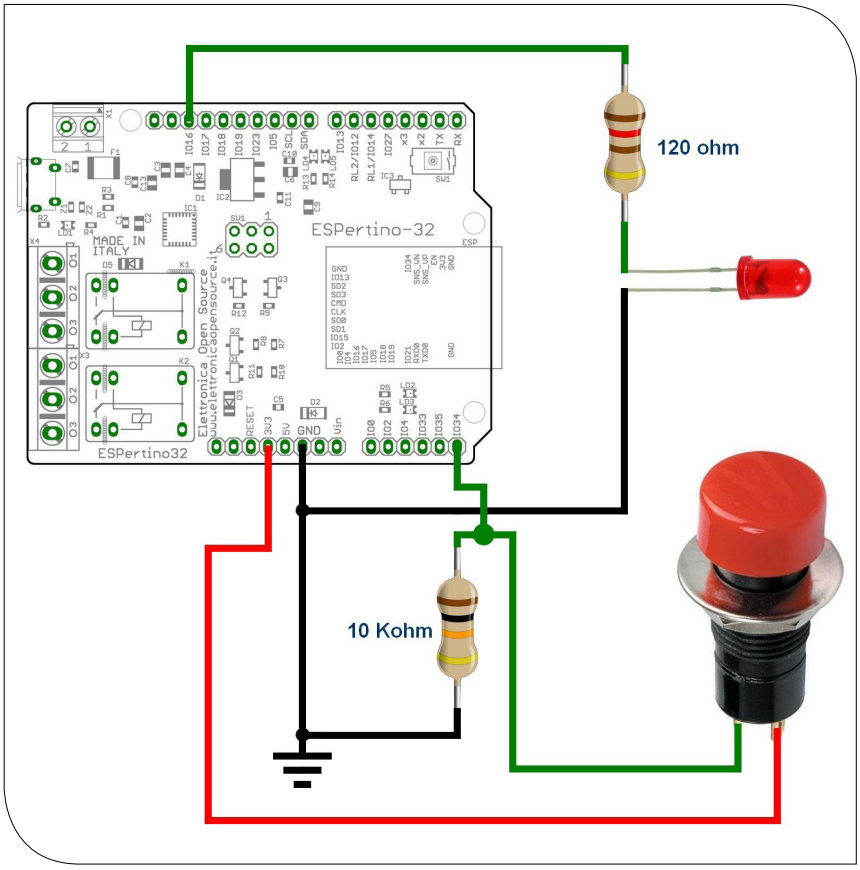



*Figura 10: il cablaggio del pulsantone salvavita*

di chiamata di emergenza verso il canale **Messenger**. Per lo scopo la stringa "url" è inizializzata all'indirizzo che innesca il servizio. Quindi, viene inoltrata la relativa richiesta di tipo HTTP, seguita dalla visualizzazione di un messaggio sul monitor seriale e da una pausa di attesa di tre secondi;

- dalla riga 36 alla riga 41 viene gestito il blocco di chiamata di emergenza verso l'**Email**. Per lo scopo la stringa "url" è inizializzata all'indirizzo che innesca il servizio. Quindi, viene inoltrata la relativa richiesta di tipo HTTP, seguita dalla visualizzazione di un messaggio sul monitor seriale e da una pausa di attesa di altri tre secondi;
- dalla riga 42 alla riga 47 viene gestito il blocco di chiamata di emergenza verso il **diario personale di Facebook**. Per lo scopo la stringa "url" è inizializzata all'indirizzo che innesca il servizio.

*Figura 11: un esempio di pulsantone da allestire quale dispositivo di chiamata d'emergenza*

Quindi, viene inoltrata la relativa richiesta di tipo HTTP, seguita dalla visualizzazione di un messaggio sul monitor seriale e da una pausa di attesa di tre secondi;

- dalla riga 48 alla riga 53 viene gestito il blocco di chiamata di emergenza verso l'**APP di IFTTT**. Per lo scopo la stringa "url" è inizializzata all'indirizzo che innesca il servizio. Quindi, viene inoltrata la relativa richiesta di tipo HTTP, seguita dalla visualizzazione di un messaggio sul monitor seriale e da una pausa di attesa di tre secondi;
- dalla riga 54 alla riga 58 viene gestito il blocco di chiamata di emergenza su un **SMS** verso un telefonino. Per lo scopo la stringa "url" è inizializzata all'indirizzo che innesca il servizio. Quindi, viene inoltrata la relativa richiesta di tipo HTTP, seguita dalla visualizzazione di un messaggio sul monitor seriale e da una pausa di attesa di tre secondi;
- dopo una pausa di un minuto (riga 62) l'intero processo si ripete e una nuova richiesta di chiamata può essere nuovamente effettuata. Il diodo Led si spegne definitivamente.

Come si nota, le chiavi personali contenute nelle URL sono state, volutamente, occultate. Nel successivo paragrafo vedremo come venire in possesso di tali chiavi.

## LA CHIAVE PER LE URL

Vedremo, adesso, dove **reperire la chiave segreta e personale**, in modo da creare alcune URL da invocare per innescare il servizio, tramite richiesta HTTP. Seguiamo le seguenti fasi:

- sul sito IFTTT cliccare su "My Applets";
- quindi cliccare su "Services";
- cliccare sul servizio "Webhooks";
- cliccare su "Settings";
- accedere alla URL proposta dalla pagina;
- usare la URL personalizzata, contenente la propria chiave e sostituendo il nome dell'evento, come visibile in figura 12.

Dal momento che i trigger creati sono cinque, le altrettanti URL sono le seguenti (con chiave camuffata ai fini dell'esempio):

https://maker.ifttt.com/trigger/**pulsantone_messenger**/with/key/XXYYXXYYXXYYXXYYXX
https://maker.ifttt.com/trigger/**pulsantone_email**/with/key/XXYYXXYYXXYYXXYYXX
https://maker.ifttt.com/trigger/**pulsantone_diario_facebook**/with/key/XXYYXXYYXXYYXXYYXX
https://maker.ifttt.com/trigger/**pulsantone_app**/with/key/XXYYXXYYXXYYXXYYXX
https://maker.ifttt.com/trigger/**pulsantone_sms**/with/key/XXYYXXYYXXYYXXYYXX

Tali indirizzi potranno anche essere inviati singolarmente, tramite un browser, per provare il loro funzionamento.

## UTILIZZO DEL PULSANTONE

Dopo aver cablato il circuito, provato i vari servizi e inscatolato professionalmente il dispositivo, si può metterlo subito all'opera, magari con la collaborazione della persona che, in seguito, dovrà utilizzarlo.

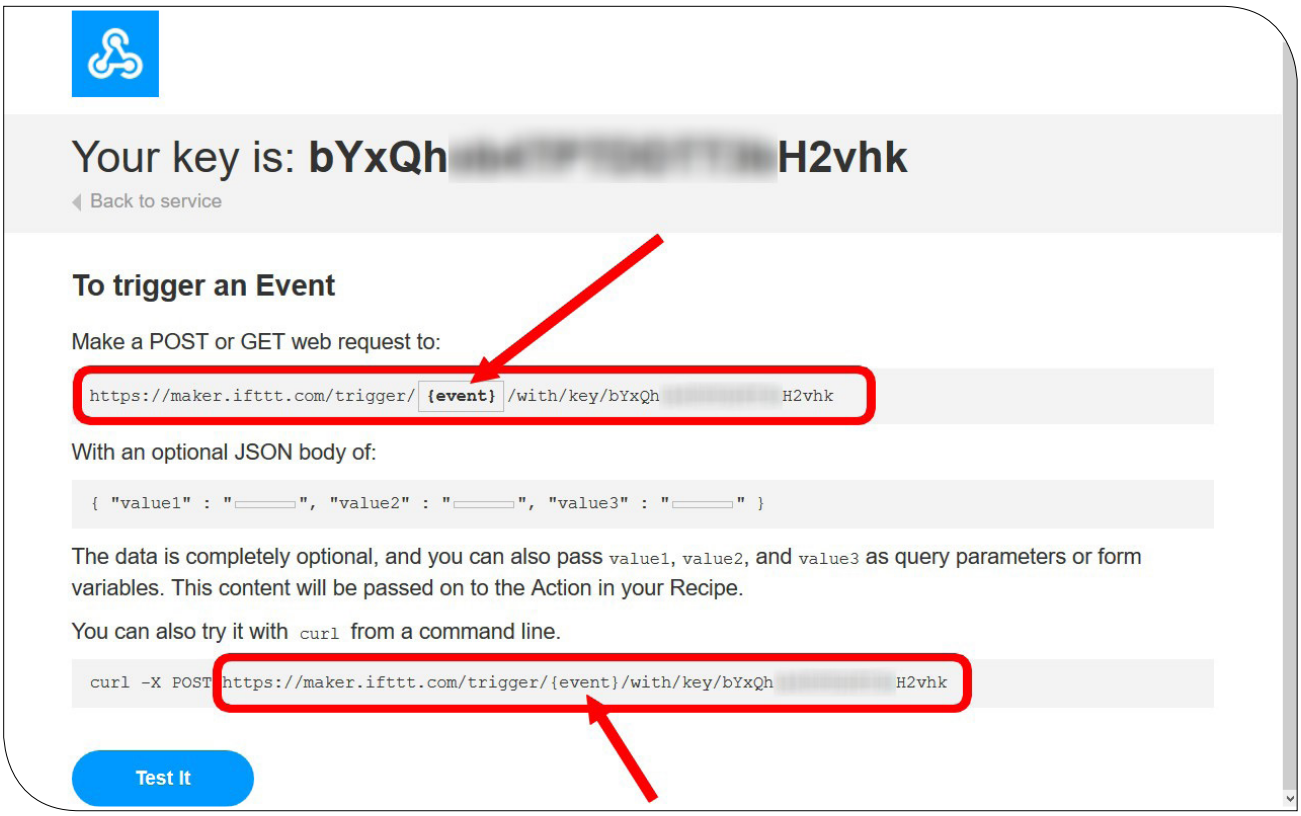



*Figura 12: come reperire la URL da utilizzare per usufruire del servizio HTTP*

*Figura 13: il monitor seriale è utile per "monitorare" le fasi del programma*

E' sufficiente, per lo scopo, posizionare il pulsantone **in una posizione facile da raggiungere** e, soprattutto, nel raggio di copertura del segnale WiFi.
Ricordiamo che per il funzionamento del dispositivo non serve il computer che è richiesto, invece, per le prove e il collaudo preliminare, al fine di leggere i messaggi sul monitor seriale (vedi figura 13).
Dopo l'accensione dell'apparecchio occorre aspettare almeno dieci secondi per dar modo di permettere la sincronizzazione del WiFi. Volutamente non sono state aggiunti altri carichi luminosi ma, per chi lo volesse, può prevedere anche un altro diodo Led per testimoniare la sincronizzazione del WiFi e dell'operatività del dispositivo stesso. Sono, ovviamente, gusti personali.
A questo punto, è sufficiente una breve o lunga pressione sul pulsantone per innescare l'allarme.

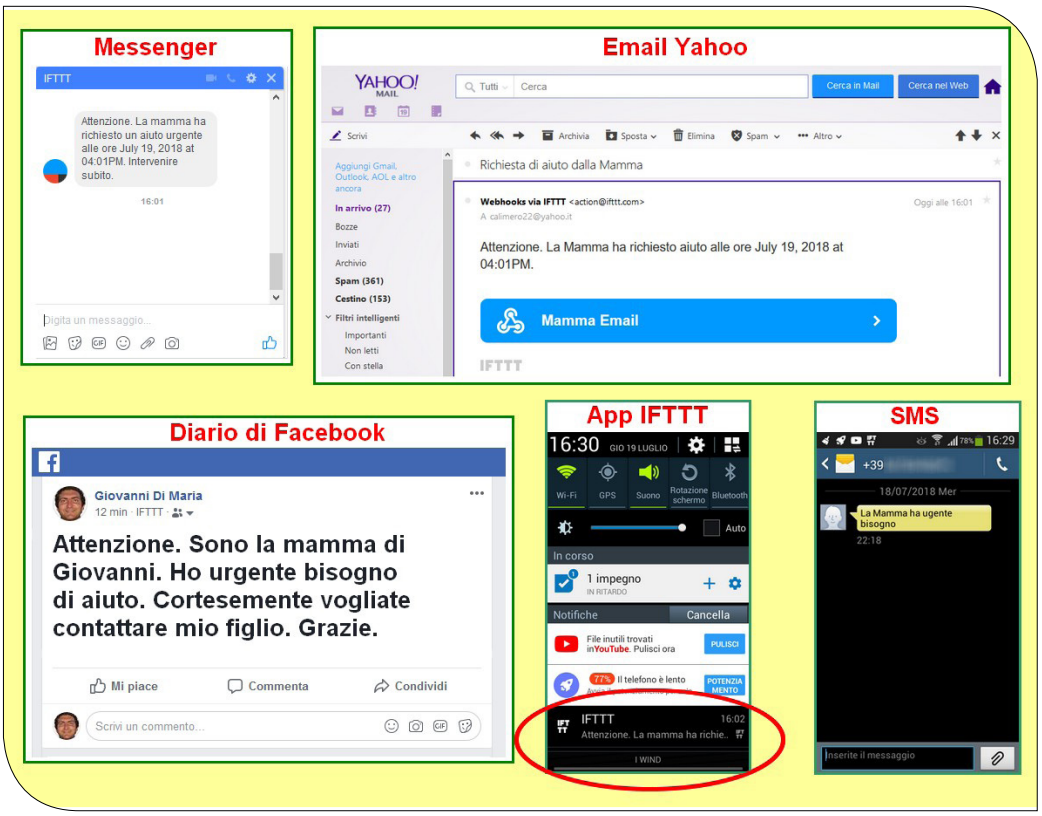



*Figura 14: i canali di comunicazione utilizzati*

Durante gli invii delle richieste di aiuto, ulteriori pressioni sul pulsante non sortiranno alcun effetto. Sequenzialmente saranno spediti dei messaggi di avvertimento alle seguenti utenze:

- messenger;
- email;
- diario di Facebook;
- applicazione di IFTTT sullo smartphone;
- SMS.

Il listato del programma è aperto a qualsiasi modifica e, proprio per questo motivo, alcuni servizi si possono eliminare dal firmware, se si pensa che la loro presenza non costituisca un fatto utile all'applicazione stessa.

**>>> Leggi anche eCall chiamata di emergenza in auto in caso di incidente [progetto completo]**

## CONCLUSIONI

Il progetto presentato nell'articolo è un utile dispositivo che si innesca non appena viene inviata una richiesta d'aiuto. La sua particolarità sta nel fatto che sono utilizzati **diversi canali di comunicazione**, come mostrato in figura 14.

Oggi, infatti, con tutti i mezzi tecnologici a disposizione, è molto difficile che una persona non venga rintracciata.

Il pulsantone va pressato solo **in caso di assoluta necessità**. Occorre dunque formare l'utente su tale aspetto, specificando che l'allarme deve essere inoltrato solo se si ha bisogno di aiuto e non per scherzo.

E' preferibile che le dimensioni del pulsante siano molto elevate, proprio per evitare di cercarlo durante le concitate fasi di emergenza, nelle quali il panico può far da padrone.

Nelle varie prove si è visto che le richieste di aiuto partono immediatamente e le notifiche (email, messaggi, ecc) arrivano a destinazione entro 10-15 secondi.

L'utilizzo del dispositivo può essere spostato leggermente, rispetto al suo fine naturale.

Ad esempio esso può essere usato come sistema anti rapina o antincendio. Anche in questo caso la fantasia e le esigenze fanno da padroni.

In ogni caso, sebbene la tecnologia offra centinaia di metodi per non far sentire sole le persone anziane (come, ad esempio le video chiamate, Internet, le telefonate, i messaggi multimediali, ecc), dovremo tutti sforzarci a essere un tantino più sensibili e cercare di colmare la solitudine dei nostri cari non tramite le più sofisticate applicazioni ma, più semplicemente, con la **nostra reale e affettuosa presenza**.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/espertino-e-ifttt-pulsantone-di-emergenza-per-anziani

# PULSANTI PER QUIZ: UN SEMPLICE PROGETTO CON ESPERTINO PER GIOCARE AI CLASSICI GIOCHI DELLA TV

di **Giovanni Di Maria**

*Li abbiamo visti centinaia di volte in TV, durante i giochi a premi, nei quali il primo concorrente che preme il pulsante acquisisce il diritto di rispondere alla domanda del conduttore. Il prototipo proposto è realizzato con la scheda ESPertino ma può essere benissimo implementato anche con altri dispositivi. Esistono diverse soluzioni per concretizzare al meglio il sistema, e alcune di esse prevedono anche l'uso degli interrupt. Vediamone alcune.*

## I QUIZ ALLA TV

Per anni abbiamo assistito, in televisione, ai divertenti e appassionanti giochi a premi pomeridiani e serali, nei quali il presentatore pone le domande a cui deve rispondere il primo concorrente che preme il proprio pulsante. Memorabili sono le puntate del giovedì sera di "Rischiatutto" ( durante gli anni 1970-1974, vedi figura 1), che teneva letteralmente incollati, davanti la TV, milioni di persone grazie alla simpatia e alla professionalità di **Mike Bongiorno**. Con molta probabilità fu lui l'ideatore di questa casistica di giochi.

## UN PULSANTE PER ACQUISIRE IL DIRITTO DI PRENOTAZIONE ALLA RISPOSTA

Il funzionamento del gioco è molto semplice: l'organizzatore propone e pone una domanda ai concorrenti, che possono essere anche alquanto numerosi. **Il primo giocatore che preme il proprio pulsante acquisisce il diritto di rispondere alla domanda, escludendo tutti gli altri**. La conquista di tale diritto è testimoniata anche da alcune segnalazioni acustiche e luminose. Tali sistemi di prenotazione possono essere utilizzati anche a casa, negli eventi organizzati con amici e parenti o nelle feste di compleanno dei bambini. Le idee possono essere tante.

## TANTE SOLUZIONI POSSIBILI

In effetti esistono tante soluzioni circuitali per implementare un sistema di prenotazione di risposta. Oggi, con i moderni microcontrollori è, realmente, un gioco da ragazzi allestire tale dispositivo. Sono sufficienti solo un paio di controlli, alcune istruzioni condizionali e decisionali e il gioco è fatto. Ma è possibile realizzare il circuito tramite la vecchia metodologia a **componenti discreti**, probabilmente più efficace e nel proseguo dell'articolo spiegheremo anche il perché. Con una quantità modesta di circuiti logici, anche tale problematica può essere brillantemente risolta, al prezzo di qualche circuito integrato aggiuntivo e una basetta elettronica più capiente.

## LA PARTE HARDWARE

A prima vista, il circuito da realizzare sembrerebbe non presentare alcun problema. In questa tipologia di applicazioni non è critica, infatti, la parte hardware ma solo il firmware e la sua concezione. Esso è composto da alcuni pulsanti, a disposizione di ciascun giocatore, da altrettanti dispositivi luminosi e da un attuatore sonoro,

*Figura 1: il quiz a premi "Rischiatutto" di Mike Bongiorno*

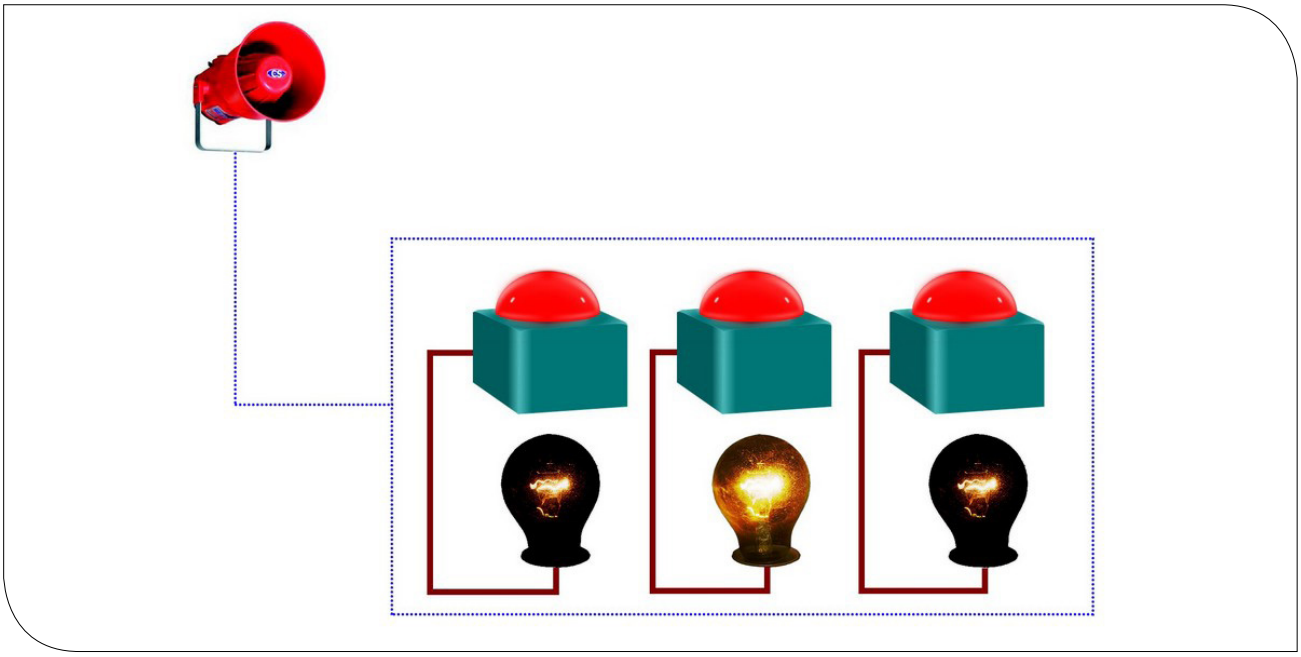


*Figura 2: i componenti base in una pulsantiera per giochi a quiz*

secondo lo schema di principio di cui alla figura 2.
Possiamo approntare, facilmente, uno schema elettrico con ESPertino, che riproduca esattamente il prototipo da realizzare. Esso è visibile in figura 3 e comprende i seguenti elementi:

- tre pulsanti normalmente aperti, uno a disposizione di ciascun concorrente;
- tre indicatori luminosi, uno per ogni giocatore;
- un unico segnalatore acustico.

Le porte, con funzionalità d'**input**, dedicate all'acquisizione dello stato logico dei pulsanti sono, rispettivamente, la GPIO33, la GPIO35 e la GPIO34. Tali porte, in stato di riposo, sono "forzate" a rilevare un livello logico basso grazie ad altrettante **resistenze di pull-down** da 10k (tale valore non è critico ed esso può essere compreso tra 1k e 1M). In caso di pressione di uno dei pulsanti lo stato logico della relativa porta si pone a livello alto (3.3V), grazie al collegamento diretto, in commutazione, con la fonte di alimentazione.

```
void setup() {
  pinMode(0, OUTPUT);  // Porta per il diodo Led
  pinMode(2, OUTPUT);  // Porta per il diodo Led
  pinMode(4, OUTPUT);  // Porta per il diodo Led
  pinMode(17, OUTPUT); // Porta per il buzzer
  pinMode(33, INPUT);  // Porta per il pulsante
  pinMode(35, INPUT);  // Porta per il pulsante
  pinMode(34, INPUT);  // Porta per il pulsante
  //----Spegne tutte le luci e anche il buzzer------
  digitalWrite(0,LOW);
  digitalWrite(2,LOW);
  digitalWrite(4,LOW);
  digitalWrite(17,LOW);
}
//-------------LOOP----------------
void loop() {
  if(digitalRead(33)==HIGH) {   // Se preme il giocatore 1.......
    digitalWrite(0,HIGH);  // Accende diodo Led
    digitalWrite(17,HIGH);  // Suona il buzzer
    delay(2000);
    digitalWrite(0,LOW);  // Spegne diodo Led
    digitalWrite(17,LOW);  // Spegne il buzzer
  }
  if(digitalRead(35)==HIGH) {   // Se preme il giocatore 2.......
    digitalWrite(2,HIGH);  // Accende diodo Led
    digitalWrite(17,HIGH);  // Suona il buzzer
    delay(2000);
    digitalWrite(2,LOW);  // Spegne diodo Led
    digitalWrite(17,LOW);  // Spegne il buzzer
  }
  if(digitalRead(34)==HIGH) {   // Se preme il giocatore 3.......
    digitalWrite(4,HIGH);   // Accende diodo Led
    digitalWrite(17,HIGH);  // Suona il buzzer
    delay(2000);
    digitalWrite(4,LOW);  // Spegne diodo Led
    digitalWrite(17,LOW);  // Spegne il buzzer
  }
}
```

```
//-------------LOOP----------------
void loop() {
  //---Legge i tre pulsanti a grandissima velocita' per-
che' le letture sono ravvicinate-----
  boolean giocatore1=digitalRead(33);
  boolean giocatore2=digitalRead(35);
  boolean giocatore3=digitalRead(34);
  //------Decide chi ha premuto prima degli al-
tri---------
  if(giocatore1==HIGH) {   // Se preme il giocatore
1.......
    digitalWrite(0,HIGH);  // Accende diodo Led
    digitalWrite(17,HIGH);  // Suona il buzzer
    delay(2000);
    digitalWrite(0,LOW);  // Spegne diodo Led
    digitalWrite(17,LOW);  // Spegne il buzzer
  }
  if(giocatore2==HIGH) {   // Se preme il giocatore
2.......
    digitalWrite(2,HIGH);  // Accende diodo Led
    digitalWrite(17,HIGH);  // Suona il buzzer
    delay(2000);
    digitalWrite(2,LOW);  // Spegne diodo Led
    digitalWrite(17,LOW);  // Spegne il buzzer
  }
  if(giocatore3==HIGH) {   // Se preme il giocatore
3.......
    digitalWrite(4,HIGH);   // Accende diodo Led
    digitalWrite(17,HIGH);  // Suona il buzzer
    delay(2000);
    digitalWrite(4,LOW);  // Spegne diodo Led
    digitalWrite(17,LOW);  // Spegne il buzzer
  }
}
```

La segnalazione dell'avvenuta prenotazione del diritto di risposta è affidata, invece, a tre diodi Led, preceduti da altrettante resistenze di limitazione da 120 ohm. Le porte utilizzate, questa volta in funzionalità di **o utput**, sono la GPIO00, la GPIO02 e la GPIO04.
Infine, la segnalazione acustica è affidata a un piccolo buzzer collegato alla porta GPIO17 configurata, anch'essa, come uscita.

## PRIMA DI TUTTO L'EQUITÀ

Perché parliamo di equità e di correttezza nella rilevazione degli eventi? Osserviamo il seguente codice per ESPertino, carichiamolo sulla scheda e collaudiamolo subito, secondo le specifiche tecniche del relativo schema elettrico.
Il listato funziona perfettamente. Dopo che il presentatore del gioco ha proposto una domanda, **il primo giocatore che preme il pulsante** acquisisce il diritto a rispondere. Questo fatto è testimoniato dall'accensione del proprio diodo Led e dalla produzione di un suono acustico, il tutto dalla durata di due secondi (tale tempo può essere, ovviamente, cambiato, secondo le esigenze del gioco). In condizioni "normali" il sistema funziona perfettamente e, addirittura, esso può essere utilizzato nelle gare ufficiali. Nel listato non è necessario prevedere alcun sistema di **gestione dell'antirimbalzo**, in quanto è importante rilevare solo la prima pressione (anche prolungata) di ogni singolo pulsante.
Facendo, però, i pignoli e i "precisini", il firmware riportato sopra potrebbe presentare qualche **piccola iniquità** nella determinazione del **primo giocatore che ha schiacciato** il pulsante. Osserviamo, allo scopo, la figura 4. Essa riporta uno stralcio del programma, e precisamente la funzione loop() del firmware.
Sappiamo tutti che, in un programma, le varie istruzioni sono eseguite una dopo l'altra, sequenzialmente. Si supponga, in un determinato momento, che l'esecuzione del listato sia **arrivata al punto rosso** del firmware. Si supponga a questo punto che, nell'arco di alcuni centesimi di secondi, il giocatore 1 prema il pulsante e, subito dopo, il giocatore 2 faccia lo stesso. Ma in questo punto del programma non c'è il controllo per il giocatore 1, in quanto è stato già sorpassato. Quindi, il giocatore 2, pur avendo premuto il pulsante leggermente dopo il giocatore 1, acquisisce il diritto a rispondere alla domanda. Parliamo, tuttavia, di **eventualità molto rare**, di episodi distanti tra loro solo pochi millesimi di secondo, ma perfettamente plausibili e verificabili.

## COME EVITARE TALE DISCRIMINAZIONE?

Il controllore ESP32 è estremamente veloce e un "giro di loop" viene eseguito **milioni di volte al secondo**. Pertanto la risoluzione temporale di rilevazione è nell'ordine dei microsecondi. Tuttavia esistono metodi che, con più accuratezza, riescono a fornire il risultato con una obiettività più elevata e a determinare con precisione maggiore il giocatore che, effettivamente, ha premuto

```
01 void setup() {
02   pinMode(0, OUTPUT);  // Porta per il diodo Led
03   pinMode(2, OUTPUT);  // Porta per il diodo Led
04   pinMode(4, OUTPUT);  // Porta per il diodo Led
05   pinMode(17, OUTPUT); // Porta per il buzzer
06   pinMode(33, INPUT);  // Porta per il pulsante
07   pinMode(35, INPUT);  // Porta per il pulsante
08   pinMode(34, INPUT);  // Porta per il pulsante
09   //----Spegne tutte le luci e anche il buzzer------
10   digitalWrite(0,LOW);
11   digitalWrite(2,LOW);
12   digitalWrite(4,LOW);
13   digitalWrite(17,LOW);
14   interrupt_SI();      // Attiva Interrupts
15 }
16 //-------------LOOP----------------
17 void loop() {
18   if(digitalRead(33) || digitalRead(35) || digitalRead(34)) {
19     interrupt_NO();       // Disattiva Interrupts
20     delay(2000);
21     digitalWrite(0,LOW);   // Spegne diodo Led
22     digitalWrite(2,LOW);   // Spegne diodo Led
23     digitalWrite(4,LOW);   // Spegne diodo Led
24     digitalWrite(17,LOW);  // Spegne il buzzer
25     interrupt_SI();       // Attiva Interrupts
26   }
27 }
28 //----Funzione richiamata quando si attiva il primo interrupt---
29 void giocatore1() {
30     digitalWrite(0,HIGH);   // Accende diodo Led
31     digitalWrite(17,HIGH);  // Suona il buzzer
32 }
33 //----Funzione richiamata quando si attiva il secondo interrupt---
34 void giocatore2() {
35     digitalWrite(2,HIGH);   // Accende diodo Led
36     digitalWrite(17,HIGH);  // Suona il buzzer
37 }
38 //----Funzione richiamata quando si attiva il terzo interrupt---
39 void giocatore3() {
40     digitalWrite(4,HIGH);   // Accende diodo Led
41     digitalWrite(17,HIGH);  // Suona il buzzer
42 }
43 //----Funzione che attiva gli interrupts----
44 void interrupt_SI() {
45   attachInterrupt(digitalPinToInterrupt(33), giocatore1, RISING );
46   attachInterrupt(digitalPinToInterrupt(35), giocatore2, RISING );
47   attachInterrupt(digitalPinToInterrupt(34), giocatore3, RISING );
48 }
49 //----Funzione che disattiva gli interrupts----
50 void interrupt_NO() {
51   detachInterrupt(digitalPinToInterrupt(33));  // Disabilita interrupt
52   detachInterrupt(digitalPinToInterrupt(35));  // Disabilita interrupt
53   detachInterrupt(digitalPinToInterrupt(34));  // Disabilita interrupt
54 }
```

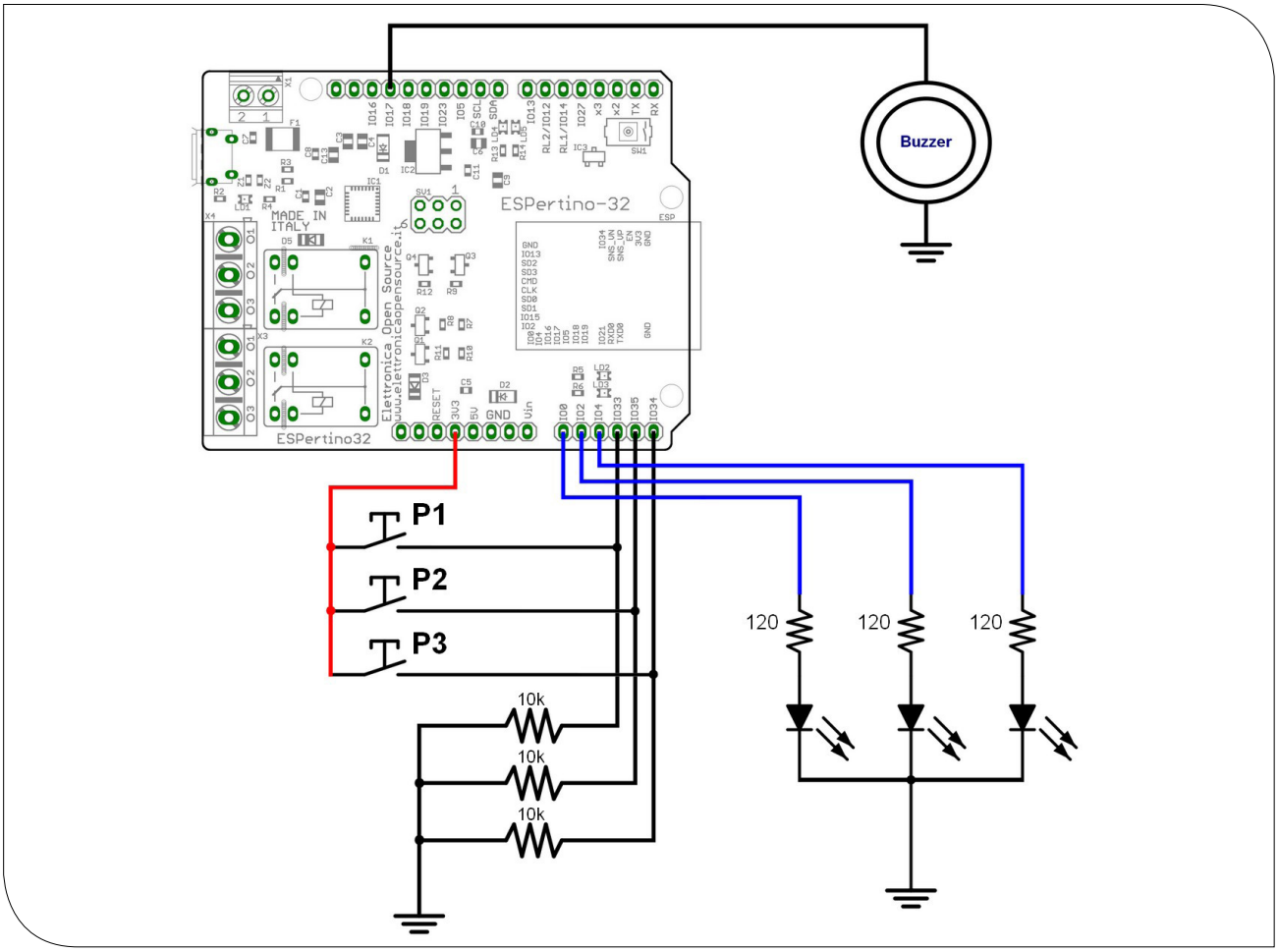



*Figura 3: schema elettrico della pulsantiera per quiz*

il proprio pulsante con anticipo su tutti. Vediamone un paio.

## METODO DELLE TRE LETTURE MOLTO RAVVICINATE

Questa soluzione non è perfetta ma offre un più alto grado di sicurezza nell'attribuzione del corretto vincitore, rispetto alla soluzione prima proposta. Essa consiste nell'effettuare le acquisizioni dei pulsanti con **altrettanti statement ravvicinati tra loro** e, poi, dedicarsi alla determinazione del giocatore più veloce nella pressione del pulsante, mediante alcune istruzioni condizionali. Il seguente listato, in cui è presente solo la funzione loop(), in quanto il setup() è lo stesso del precedente, provvede a effettuare la lettura dello stato logico dei tre pulsanti, a grandissima velocità. Quindi, "con relativa calma", decide quale è stato il giocatore a premere per primo.

## METODO DEGLI INTERRUPT

La soluzione proposta **è una delle più critiche** che esistano nella programmazione, in quanto utilizza la tecnica degli **interrupt**. Essi permettono di attivare automaticamente alcune funzionalità in presenza di eventi ben precisi sull'hardware, senza effettuare noiose e ripetitive istruzioni condizionali nel programma (polling), come ben evidenziato dalla figura 5. Ben lungi dallo spiegare, in questa sede, cosa siano gli interrupt, proponiamo il listato di seguito, per poi spiegarlo in dettaglio.

Le righe di programma sono state volutamente numerate in modo progressivo solo per dare un riferimento per i relativi commenti. In fase di digitazione nell'IDE i numeri, ovviamente, non devono essere inseriti:

- le righe di programma comprese tra 1 e 15 contengono la funzione **setup()**, come nei precedenti listati, nella quale vengono preparate le porte di I/O alle rispettive funzionalità. Alla riga 14 viene chiamata la funzione interrupt_SI() che ha il compito di predisporre i segnali d'interrupt sul **fronte di salita** delle porte 33, 34 e 35, che fanno capo ai rispettivi pulsanti;
- le righe di programma comprese tra 16 e 27

```
void loop() {
  if(digitalRead(33)==HIGH) {   // Se preme il giocatore 1.......
    digitalWrite(0,HIGH);  // Accende diodo Led
    digitalWrite(17,HIGH);  // Suona il buzzer
    delay(2000);
    digitalWrite(0,LOW);  // Spegne diodo Led
    digitalWrite(17,LOW);  // Spegne il buzzer
  }
  if(digitalRead(35)==HIGH) {   // Se preme il giocatore 2.......
    digitalWrite(2,HIGH);  // Accende diodo Led
    digitalWrite(17,HIGH);  // Suona il buzzer
    delay(2000);
    digitalWrite(2,LOW);  // Spegne diodo Led
    digitalWrite(17,LOW);  // Spegne il buzzer
  }
  if(digitalRead(34)==HIGH) {   // Se preme il giocatore 3.......
    digitalWrite(4,HIGH);   // Accende diodo Led
    digitalWrite(17,HIGH);  // Suona il buzzer
    delay(2000);
    digitalWrite(4,LOW);  // Spegne diodo Led
    digitalWrite(17,LOW);  // Spegne il buzzer
  }
}
```

*Figura 4: una piccola iniquità nel listato*

contengono la funzione **loop()**. Essa controlla, ciclicamente, se un diodo led relativo a un concorrente si è illuminato, nel qual caso disabilita momentaneamente la gestione degli interrupt, predispone una pausa di 2 secondi per mantenere le segnalazioni luminose e sonore attive, per poi disattivarle e infine riattiva la gestione degli interrupt;

- le righe di programma comprese tra 28 e 32 contengono la funzione **giocatore1()**, richiamata automaticamente al verificarsi dell'interrupt causato dal primo giocatore, in caso di pressione del pulsante. In essa avviene l'attivazione del diodo Led del giocatore e del segnalatore acustico;
- le righe di programma comprese tra 33 e 37 contengono la funzione **giocatore2()**. In essa avviene l'attivazione del diodo Led del giocatore e del segnalatore acustico, richiamata automaticamente al verificarsi dell'interrupt causato dal secondo giocatore, in caso di pressione del pulsante;
- le righe di programma comprese tra 38 e 42 contengono la funzione **giocatore3()**. In essa avviene l'attivazione del diodo Led del giocatore e del segnalatore acustico, richiamata automaticamente al verificarsi dell'interrupt causato dal terzo giocatore, in caso di pressione del pulsante;
- le righe di programma comprese tra 43 e 48 contengono la funzione **interrupt_SI()**. Essa si occupa della predisposizione e della configurazione degli interrupt, secondo la seguente sintassi:
  - **attachInterrupt(Interrupt, func_chiamata, modalità)**;
- le righe di programma comprese tra 49 e 54 contengono la funzione **interrupt_NO()**, che disattiva tutti gli interrupt, all'occorrenza. Se, infatti, un giocatore ha premuto per primo il proprio pulsante e le segnalazioni sono attive, gli altri concorrenti non dovranno avere la facoltà di prenotarsi per la risposta.

Le varie modalità di discriminazione della variazione di segnale, su una porta d'ingresso, previste dalla funzione attachInterrupt(), sono le seguenti:

- **LOW**: attiva l'interrupt se lo stato logico del pin è basso;
- **CHANGE**: attiva l'interrupt quando lo stato logico del pin cambia;
- **RISING**: attiva l'interrupt se lo stato logico del pin cambia da basso ad alto;
- **FALLING**: attiva l'interrupt se lo stato logico del pin cambia da alto a basso;
- **HIGH**: attiva l'interrupt se lo stato logico del pin è alto.

**>>> Leggi anche: Interrupt Vs polling: qual è la differenza?**

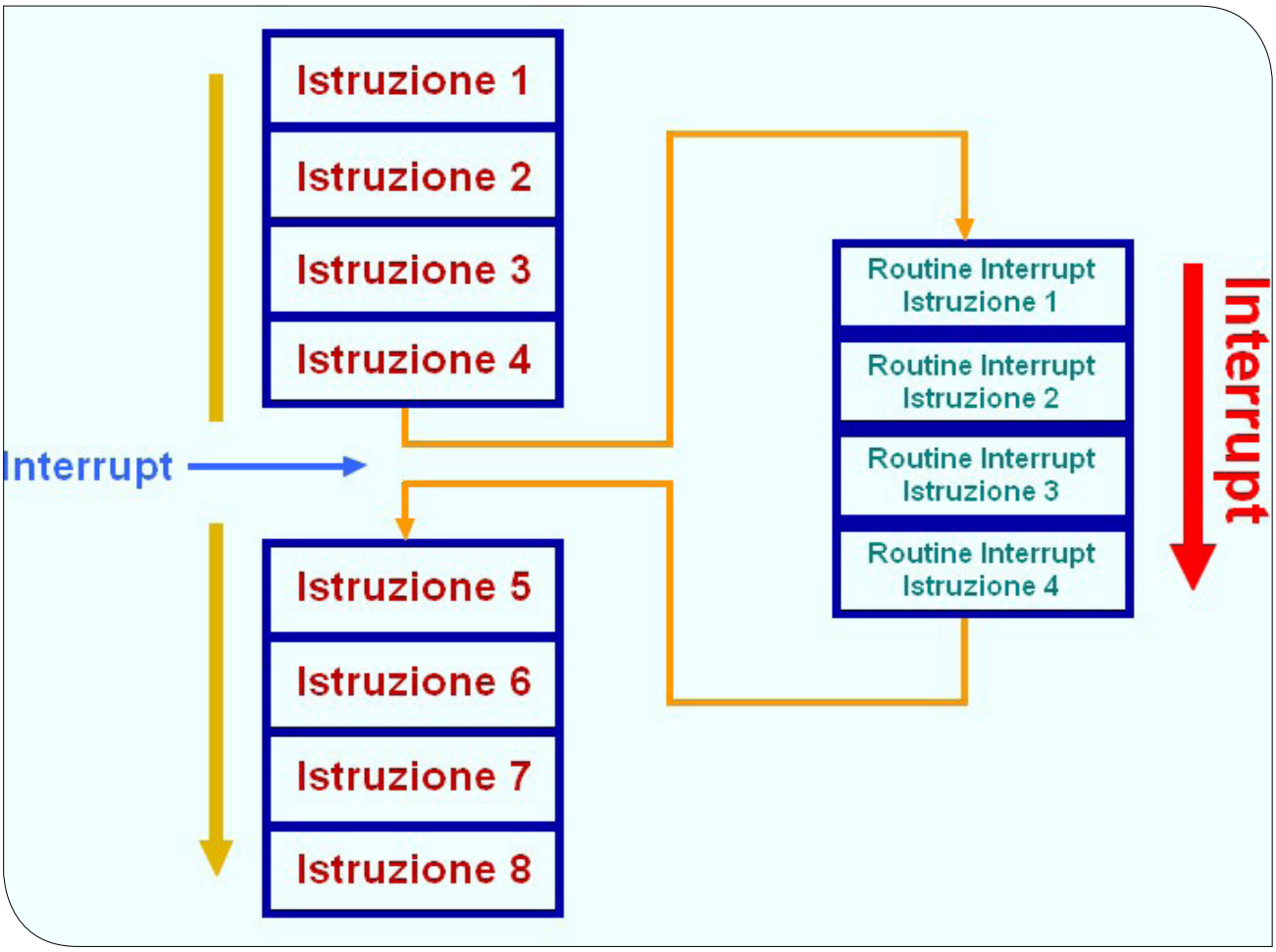



*Figura 5: la filosofia di funzionamento degli interrupt*

## CONSIGLI PER L'USO DEGLI INTERRUPT

L'utilizzo degli interrupt rende il codice eseguibile **estremamente critico** e la ricerca di eventuali errori potrebbe risultare alquanto complessa e difficoltosa. Pertanto essi dovrebbero essere utilizzati sono quando è strettamente necessario. Ecco alcuni consigli che il programmatore dovrebbe tenere a mente quando progetta un sistema che faccia uso degli interrupt:

- nella funzione richiamata da un interrupt, il **delay()** non funziona;
- la funzione dovrebbe durare il meno tempo possibile ed essere formata da pochissime istruzioni;
- durante la chiamata di un interrupt la comunicazione seriale non è affidabile;
- dentro una funzione chiamata, gli interrupt dovrebbero essere disabilitati momentaneamente, per poi riattivarli all'uscita. In caso contrario potrebbe innescarsi un processo di ricorsione anomala e critica, molto difficile da correggere;
- sempre all'interno della funzione chiamata, si devono usare le **variabili volatili**.

## CONCLUSIONI

La scheda ESPertino ci è stata utile anche in questa occasione, soprattutto a livello didattico. Quella presentata è stata, ovviamente, un'applicazione estremamente semplice ma che mostra come, **per ogni problema, esistono diverse soluzioni**. Si invitano i lettori a cercarne delle altre e a condividere pensieri e osservazioni nei commenti dedicati. Augurandovi una buona realizzazione a tutti, ci congediamo da voi come usava fare il grande Mike Bongiorno, con l'affettuoso e gioioso augurio: "Allegria".

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/pulsanti-per-quiz-un-semplice-progetto-con-espertino-per-giocare-ai-classici-giochi-della-tv

# APPLICAZIONI CRITTOGRAFICHE DI ESPERTINO

di **Stefano Lovati**

*La scheda ESPertino presenta tutte le caratteristiche richieste per supportare anche le più critiche e sfidanti applicazioni in ambito IoT (Internet of Things). Oltre alla connettività, resa possibile grazie alle interfacce integrate WiFi e Bluetooth, ESPertino dispone anche di funzionalità avanzate per la crittografia, con acceleratori hardware compatibili con i più diffusi algoritmi crittografici*

## INTRODUZIONE

Il modulo **ESP32** che equipaggia la board **ESPertino** dispone di acceleratori hardware integrati capaci di supportare diversi algoritmi crittografici. Come visibile nello schema a blocchi di **Figura 1**, l'accelerazione crittografica hardware comprende i seguenti moduli: **AES**, **SHA-2** (con dimensione dell'hash pari a 256 bit), **RSA** e **ECC**. A questi algoritmi di crittografia (che potremmo definire come "classici") si accompagna un generatore di numeri casuali (**RNG**, acronimo di Random Number Generator), particolarmente utile per la creazione di chiavi per la cifratura/decifrazione dei dati.

L'acceleratore **AES** (acronimo di Advanced Encryption Standard) supporta sei algoritmi conformi alle specifiche FIPS PUB 197, in particolare: AES-128, AES-192 e AES-256, sia per la crittografia che per la decrittografia. Le prestazioni dell'acceleratore AES sono notevolmente superiori rispetto a quelle degli stessi algoritmi im-

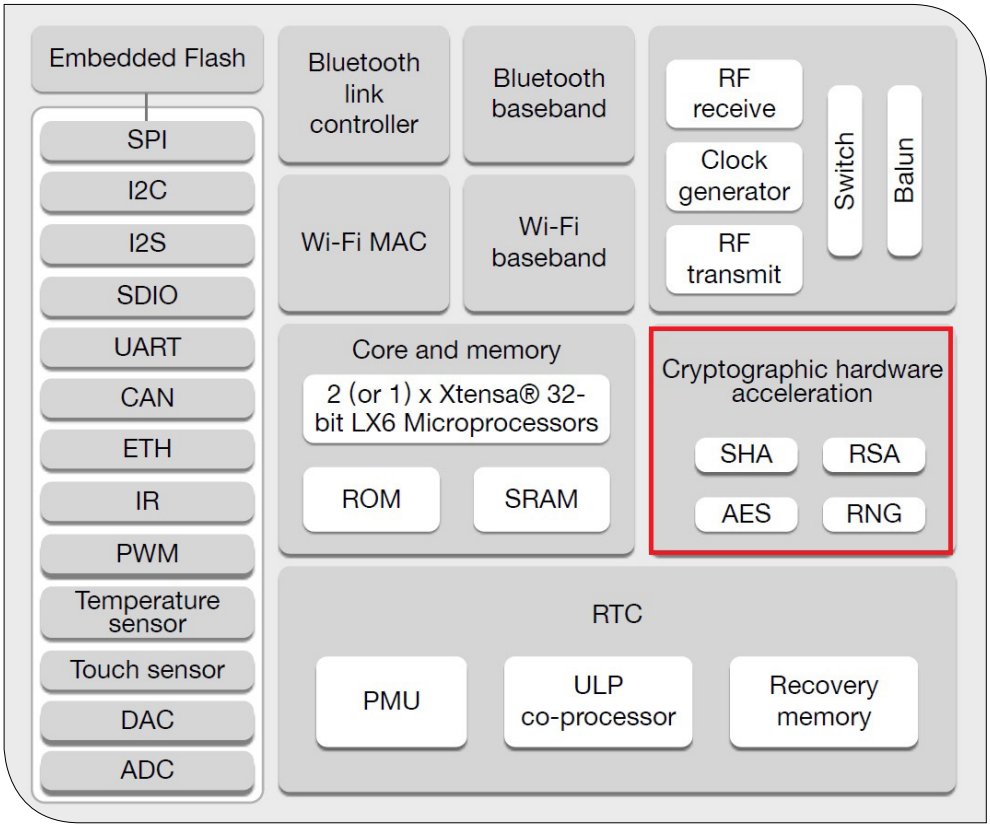



*Figura 1: i blocchi che compongono l'acceleratore hardware per la crittografia integrato nel modulo ESP32*

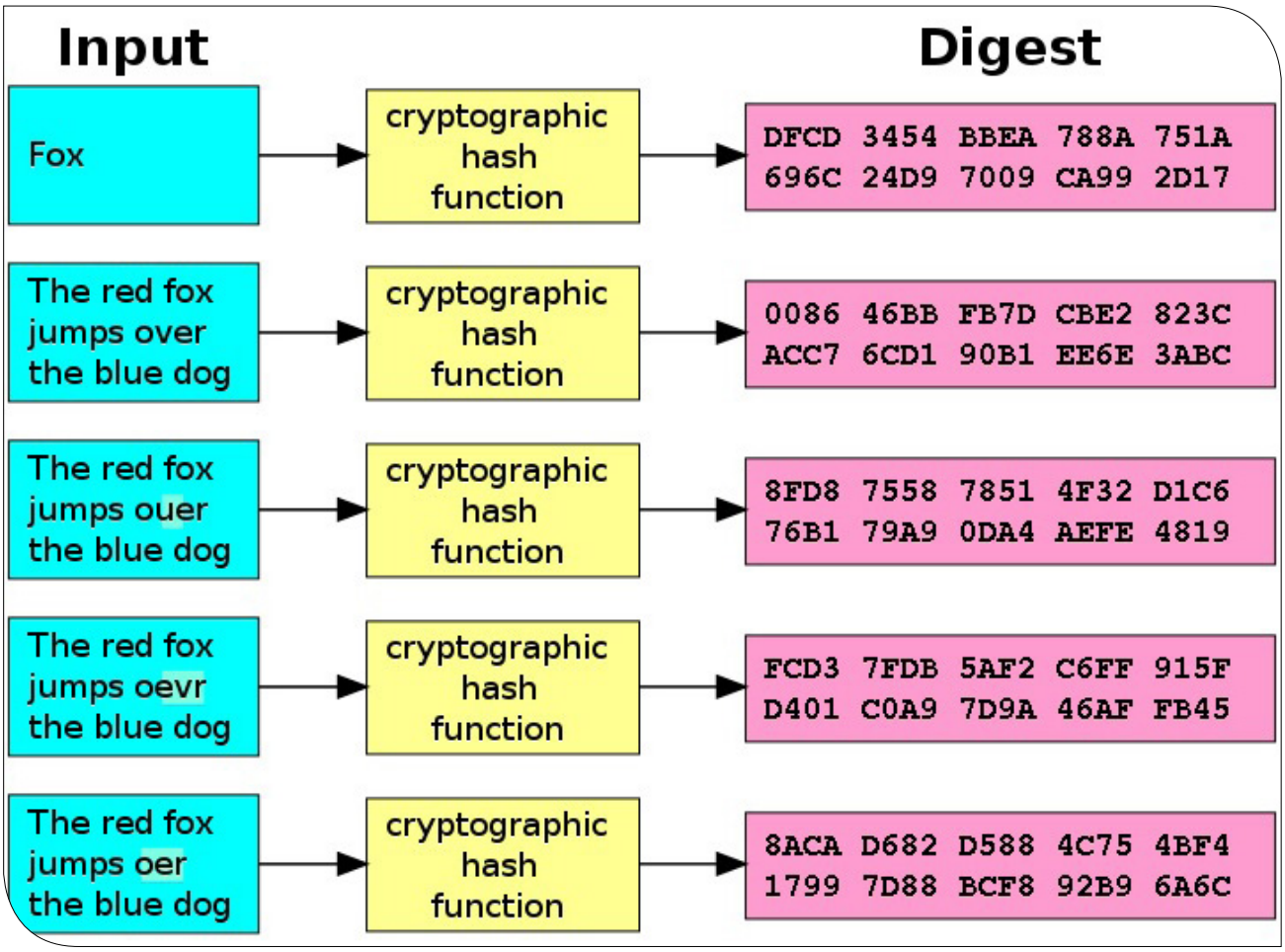



*Figura 2: rappresentazione schematizzata di una funzione crittografica di hash*

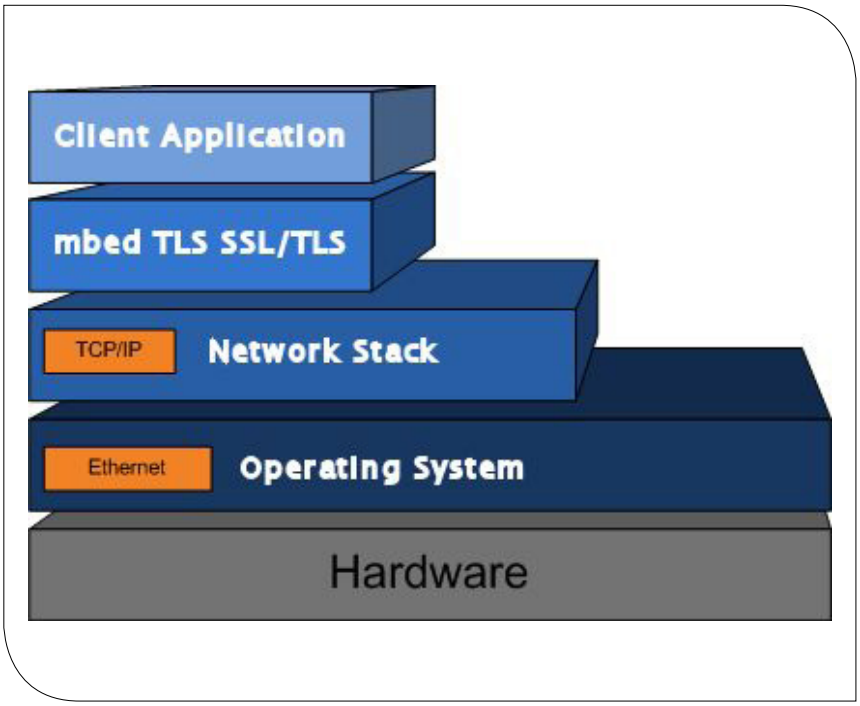



*Figura 3: posizionamento di mbed TLS all'interno degli strati software di una generica applicazione embedded*

plementati a livello software. Basti pensare che sono richiesti da 11 a 15 cicli di clock per criptare un singolo messaggio e da 21 a 22 cicli di clock per eseguire la decodifica dello stesso messaggio. L'acceleratore **SHA** supporta quattro algoritmi conformi alle specifiche FIPS PUB 180-4, in particolare: SHA-1, SHA-256, SHA-384 e SHA-512. Gli ultimi tre algoritmi appartengono al gruppo di funzioni di hash **SHA-2**, con dimensione dell'hash pari a, rispettivamente, 256, 384 e 512 bit. L'acceleratore **RSA** fornisce invece un supporto hardware per le operazioni aritmetiche a multipla precisione utilizzate negli algoritmi di cifratura asimmetrica RSA. L'algoritmo RSA (acronimo di Rivest Shamir Adelman) vine largamente

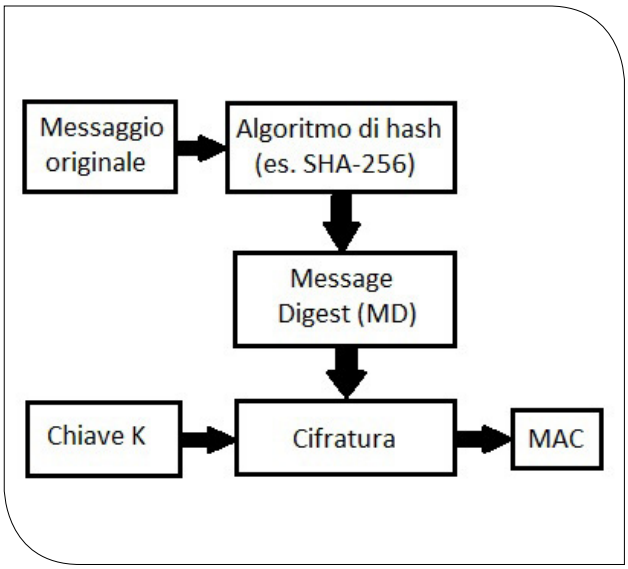



*Figura 4: schema di principio del processo di generazione del codice HMAC*

utilizzato per la cifratura di chiavi pubbliche. Infine, il modulo ESP32 include un **generatore di veri numeri casuali**, utilizzabile come base per molte operazioni crittografiche. I numeri casuali sono generati basandosi sull'entità del **rumore presente nella componente a radio frequenza** del modulo, ovvero nei transceiver Bluetooth e WiFi. Quando le interfacce WiFi e Bluetooth sono disabilitate, il generatore produce dei numeri pseudo casuali. Viceversa, quando le stesse interfacce sono abilitate, il generatore è alimentato da due bit di entropia ad ogni ciclo di clock del bus APB (normalmente impostato a 80 MHz). Ne consegue che, nel caso di massima entropia, è conveniente leggere il registro del generatore random a una frequenza massima di 5 MHz (il registro è a 32 bit, 2 bit vengono scritti ad ogni ciclo di clock APB). Dopo questa breve, ma doverosa, panoramica sui moduli hardware per la crittografia integrati nel modulo ESP32, ci poniamo l'obiettivo di applicare e testare uno degli acceleratori hardware maggiormente utilizzato a livello pratico: lo **SHA-256**.

## SHA-256

In altri articoli relativi al tema criptovalute e blockchain abbiamo già trattato l'argomento relativo alle funzioni di hash e in particolare a una loro categoria molto importante: le **funzioni crittografiche di hash**. Ci interessa ora solo ricordare che tali funzioni ricevono in ingresso un qualsivoglia blocco dati (messaggio) di lunghezza arbitraria, applicano allo stesso un determinato algoritmo e producono in uscita una sequenza di bit di dimensione

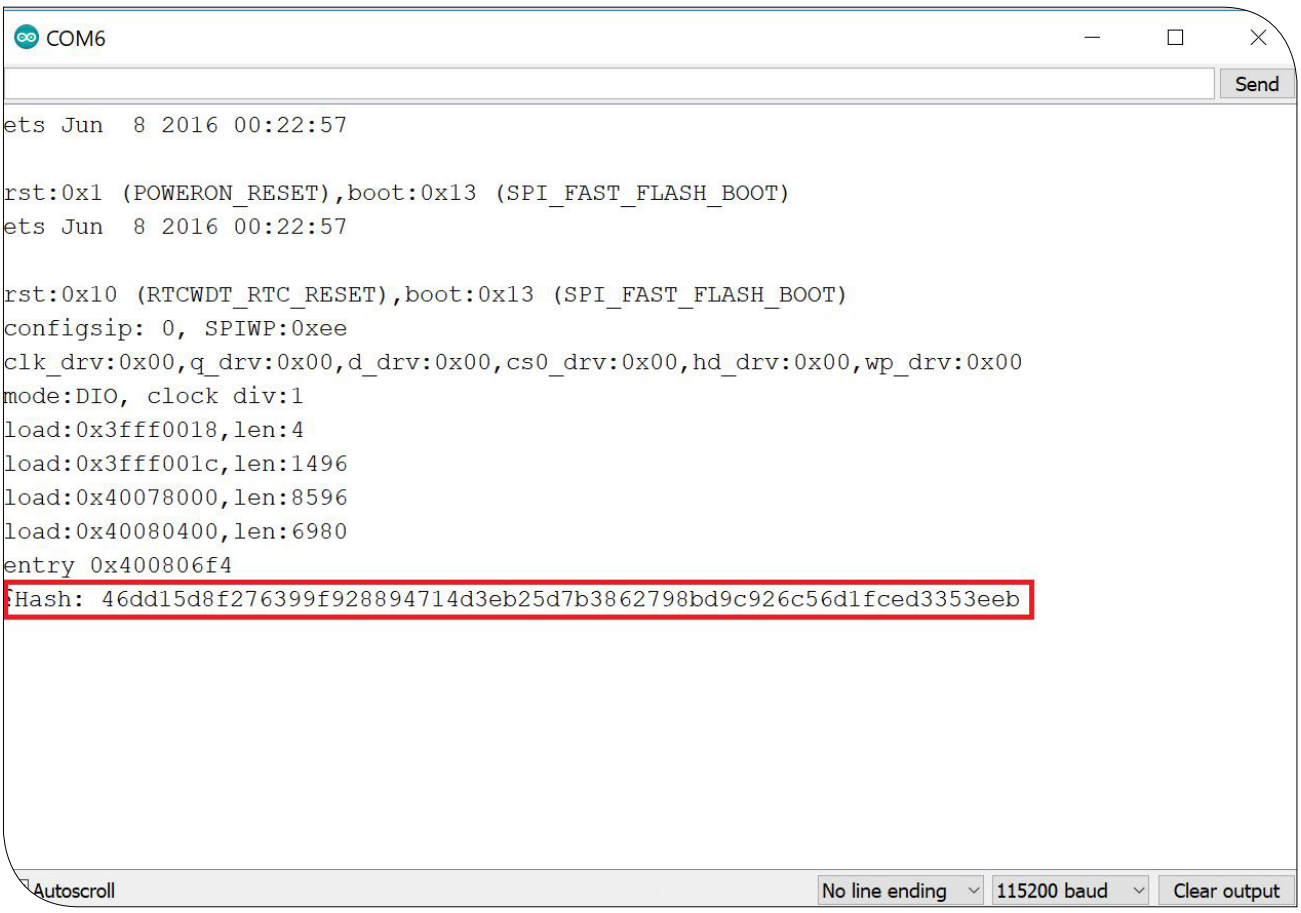



*Figura 5: trace su linea seriale prodotto dall'esecuzione dello sketch su ESPertino*

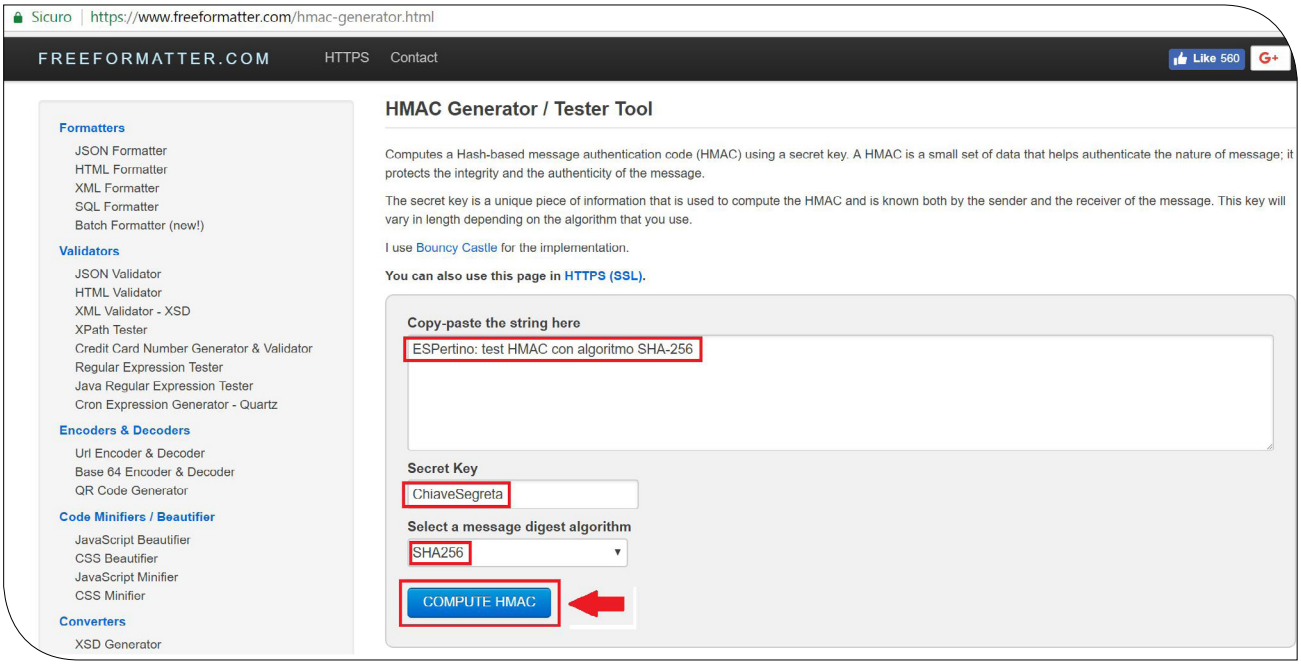



*Figura 6: tool disponibile online per la verifica del codice HMAC prodotto da ESPertino*

fissa (indicato spesso con il termine *digest* o semplicemente **hash**). Il comportamento di una generica funzione crittografica di hash è schematizzato nella **Figura 2**. **SHA-256** è un algoritmo di hash e come tale definisce le operazioni necessarie per implementare una determinata funzione crittografica di hash. In particolare, SHA-256 appartiene all'insieme di algoritmi **SHA-2**, ovvero alla versione 2 del Secure Hash Algorithm, un insieme di specifiche redatto dall'ente statunitense **NIST** (National Institute of Standards and Technology). A differenza di altri algoritmi precedenti (come lo **SHA-1** e l'**MD5**), lo SHA-2 non è ancora stato "rotto", intendendo con ciò la possibilità di trovare (applicando tipicamente il metodo della "forza bruta" o altre sue varianti) un messaggio di ingresso all'algoritmo che produca un hash in uscita esattamente identico a un valore desiderato (equivale più o meno a trovare la combinazione per aprire una cassaforte..). Lo SHA-256 rappresenta pertanto un'ottima garanzia di sicurezza ed è questo il motivo per cui viene largamente utilizzato per proteggere le transazioni generate con le criptovalute. A titolo di esempio, ricordiamo che la rete su cui si basa la principale criptovaluta (la blockchain del Bitcoin) utilizza proprio l'algoritmo SHA-256 per implementare la Proof of Work, ovvero il processo matematico che porta alla validazione delle transazioni presenti nella blockchain.

## IL PROGETTO

Lo scopo del progetto (interamente software) presentato in questo articolo è quello di **dimostrare le capacità crittografiche della scheda ESPertino** mettendo alla prova l'acceleratore hardware che supporta l'algoritmo SHA-256. In particolare, calcoleremo l'hash di un messaggio utilizzando l'algoritmo **SHA-256**, verificando poi la correttezza del valore (*digest*) prodotto in uscita confrontandolo con il valore calcolato da uno dei tanti generatori di SHA-256 disponibili online. È importante sottolineare come questo esperimento non sia fine a se stesso: i protocolli di rete "sicuri" (come ad esempio **HTTPS** o quelli basati su **SSH/SSL**, tanto per citarne un paio) si basano proprio su funzioni crittografiche di hash per cifrare i messaggi trasmessi e forniscono pertanto una valida protezione nei confronti degli hacker o degli "sniffer" presenti in rete. Per tutte le applicazioni e i dispositivi che operano in ambito **IoT** è oggi fondamentale disporre di **meccanismi collaudati e affidabili** atti a garantire un elevato grado di **sicurezza** a più livelli. Occorre perciò evitare di trasmettere le informazioni "in chiaro" (utilizzando ad esempio un protocollo non protetto come l'HTTP) e utilizzare solo comunicazioni cifrate. Non è un caso se **MQTT** (uno dei principali protocolli utilizzati nelle applicazioni IoT e M2M) utilizza il protocollo crittografico TLS/SSL per la comunicazione sicura end to end. Per ragioni di semplicità, il programma sarà sviluppato con l'ambiente **Arduino IDE**, opportunamente configurato con il plugin "Arduino core for the ESP32" disponibile su GitHub. Per supportare il processo software di cifratura e decifratura dei messag-

```
#include <mbedtls/md.h>

void setup()
{
  Serial.begin(115200);

  char *key = “ChiaveSegreta”;
  char *payload = “ESPertino: test HMAC con algoritmo SHA-256”;
  byte hmacResult[32];

  mbedtls_md_context_t ctx;
  mbedtls_md_type_t md_type = MBEDTLS_MD_SHA256;

  const size_t payloadLength = strlen(payload);
  const size_t keyLength = strlen(key);

  mbedtls_md_init(&ctx);
  mbedtls_md_setup(&ctx, mbedtls_md_info_from_type(md_type), 1);
  mbedtls_md_hmac_starts(&ctx, (const unsigned char *) key, keyLength);
  mbedtls_md_hmac_update(&ctx, (const unsigned char *) payload, payloadLength);
  mbedtls_md_hmac_finish(&ctx, hmacResult);
  mbedtls_md_free(&ctx);

  Serial.print(“Hash: “);

  for (int i=0; i<sizeof(hmacResult); i++)
  {
    char str[3];

    sprintf(str, “%02x”, (int)hmacResult[i]);
    Serial.print(str);
  }
}

void loop()
{
}
```

*Figura 7: l'ouput prodotto dal tool di verifica online coincide con quello prodotto da ESPertino*

gi utilizzeremo poi una libreria open source largamente impiegata nelle applicazioni embedded di vario tipo: **mbed TLS**.

## LA LIBRERIA MBED TLS

Inizialmente nota come PolarSSL, mbed TLS permette agli sviluppatori di **aggiungere funzionalità crittografiche** (in particolare, il supporto al protocollo sicuro SSL/TLS) all'interno dei propri **progetti embedded**, con uno sforzo implementativo minimo. Come visibile in **Figura 3**, la libreria si inserisce tra il livello corrispondente allo stack di rete TCP/IP e il livello dell'applicazione. Uno dei vantaggi di questa libreria è quello di essere facilmente portabile su più piattaforme tra cui, appunto, il modulo ESP32. Inoltre, il plugin Arduino Core menzionato in precedenza integra già nativamente tale libreria, che è quindi immediatamente disponibile senza dover installare librerie aggiuntive.

## HMAC

L'applicazione che ora descriveremo è di fatto un'applicazione **HMAC** (acronimo di keyed Hash Message Authentication Code): utilizza cioè una modalità per l'autenticazione dei messaggi basata sulle funzioni di hash. I sistemi di autenticazione HMAC permettono di verificare sia l'integrità di un messaggio che la sua autenticità e possono essere utilizzati a priori con qualunque tipo di funzione crittografica di hash. Nel nostro caso utilizzeremo l'algoritmo **SHA-256** e, pertanto, il codice HMAC prodotto avrà dimensione pari a 32 byte (256 bit). Lo schema di principio relativo alla generazione del codice HMAC è visibile in **Figura 4**. Il mittente del messaggio genera il codice HMAC e lo invia insieme allo stesso messaggio. Il ricevente può generare a sua volta il codice a partire dal messaggio ricevuto e verificare se il valore calcolato corrisponde a quello ricevuto. Come detto in precedenza, HMAC viene utilizzato per verificare l'autenticità del messaggio, operazione resa possibile dalla presenza di una chiave condivisa (indicata con K in **Figura 4**) tra i due nodi posti in comunicazione.

Le prime implementazioni di HMAC utilizzavano algoritmi di cifratura poi rivelatosi deboli, come lo **SHA-1** e l'**MD5** (MD è proprio l'acronimo di Message Digest). Oggi si tende a utilizzare implementazioni più sicure, come ad esempio lo SHA-256 che abbiamo scelto per la nostra applicazione. La robustezza del meccanismo HMAC dipende fondamentalmente da due fattori:

- lunghezza della chiave utilizzata per cifrare i messaggi: le chiavi "lunghe" sono più sicure di quelle "corte";
- modalità con cui sono generate le chiavi: il grado di sicurezza maggiore si ottiene con chiavi generate in modo casuale.

La protezione da attacchi tipici come *man in the middle* ("uomo nel mezzo") è garantita dalla presenza della chiave: se ad esempio un hacker si inserisce nella rete modificando il messaggio, il ricevente genererà un codice HMAC diverso da quello ricevuto e pertanto scarterà il messaggio. Inoltre, l'hacker non conosce la chiave e pertanto non sarà in grado di contraffare il messaggio e generare un nuovo codice valido.

## LO SKETCH

Analizziamo ora la parte software del progetto che, come detto in precedenza, è composta da uno sketch sviluppato in ambiente Arduino IDE. La prima cosa da fare è includere il file header (**md.h**) della libreria mbed TLS, il quale contiene i prototipi delle funzioni esposte dalla libreria stessa.

```
#include <mbedtls/md.h>
```

Occorre poi implementare la funzione setup() dello sketch, partendo dalla configurazione della linea seriale che utilizzeremo per visualizzare il risultato prodotto dall'operazione HMAC:

```
Serial.begin(115200);
```

Dobbiamo poi definire i due parametri in ingresso alla funzione di hash: il messaggio a cui vogliamo applicare la codifica e la chiave segreta.

```
char *key = “ChiaveSegreta”;
 char *payload = “ESPertino: test HMAC con algoritmo SHA-256”;
```

Avremo anche bisogno di allocare un **array di byte** per contenere il codice di autenticazione generato dall'applicazione del processo HMAC. Poichè la funzione di hash SHA-256 produce un hash di 256 bit, ci servirà un array con dimensione esattamente pari a **32 byte**.

```
byte hmacResult[32];
```

Le funzioni della libreria mbed TLS utilizzano una particolare struttura dati (il “**contesto**”, o *context*), utilizzata per mantenere lo stato interno tra chiamate successive a funzioni della libreria stessa. La variabile utilizzata dallo sketch per mantenere questo contesto è **ctx**.

```
mbedtls_md_context_t ctx;
```

Tra i parametri che occorre passare alla funzione di hash è presente il tipo di algoritmo di hash che si vuole utilizzare, definito dal tipo enumerato *mbedtls_md_type_t*. La corrispondente variabile dello sketch è **md_type**. Nel nostro caso, poiché vogliamo utilizzare l'algoritmo di hash SHA-256, assegneremo a tale variabile il valore enumerato **MBEDTLS_MD_SHA256**.

```
mbedtls_md_type_t md_type = MBEDTLS_MD_SHA256;
```

Poiché le funzioni della libreria mbed TLS dovranno ricevere come parametro sia la lunghezza del messaggio che quella della chiave, dovremo memorizzare tali valori in due apposite variabili. Trattandosi in entrambi i casi di stringhe, la lunghezza del messaggio e della chiave verrà ottenuta tramite chiamata alla funzione di libreria *strlen*.

```
const size_t payloadLength = strlen(payload);
 const size_t keyLength = strlen(key);
```

Siamo a questo punto pronti per chiamare le funzioni della libreria mbed TLS. La prima chiamata da eseguire è quella relativa alla funzione **mbedtls_md_init**, la quale riceve in ingresso un puntatore a una variabile di tipo **mbedtls_md_context_t**. La funzione ha lo scopo di inizializzare il contesto precedentemente menzionato e, pertanto, dovremo passare come parametro l'indirizzo della variabile **ctx**. Si noti come la funzione non restituisca alcun valore (valore di ritorno di tipo *void*).

```
mbedtls_md_init(&ctx);
```

Successivamente dovremo chiamare la funzione **mbedtls_md_setup**, il cui compito è selezionare la funzione di hash da utilizzare e allocare le strutture dati interne richieste. Il primo argomento è un puntatore a una variabile di tipo **mbedtls_md_context_t**, mentre il secondo argomento è una variabile di tipo **mbedtls_md_info_t**, utilizzata per contenere alcune informazioni relative alla funzione di hash. Per ottenere questa variabile occorre chiamare la funzione **mbedtls_md_info_from_type**, passando in ingresso la variabile di tipo **mbedtls_md_type_t** precedentemente dichiarata.
Ritornando alla funzione **mbedtls_md_setup**, notiamo come esista un terzo parametro (di tipo intero) che specifica se si intende utilizzare l'HMAC (parametro uguale a 1) oppure no (parametro uguale a 0).
Nel nostro caso eseguiamo la chiamata di funzione con il terzo parametro posto a 1, dal momento che vogliamo utilizzare la funzionalità HMAC.
In caso di successo la funzione restituisce il valore 0, altrimenti viene restituito uno tra i due seguenti codici di errore:

- **MBEDTLS_ERR_MD_BAD_INPUT_DATA** se uno o più dei parametri di ingresso non è valido;
- **MBEDTLS_ERR_MD_ALLOC_FAILED** se si è verificato un problema durante l'allocazione della memoria.

Entrambi questi codici di errore sono definiti nel file header **md.h**. Per ragioni di semplicità, supporremo che la chiamata di funzione vada sempre a buon fine e pertanto non testeremo il valore di ritorno.

```
mbedtls_md_setup(&ctx, mbedtls_md_info_from_type(md_type), 1);
```

Successivamente dobbiamo eseguire una chiamata alla funzione **mbedtls_md_hmac_starts**, il cui compito

è impostare la chiave HMAC e prepararsi all'autenticazione del messaggio. Oltre a un puntatore alla variabile contesto, la funzione riceve come secondo parametro la chiave e come terzo parametro la lunghezza della chiave. Il valore di ritorno è 0 in caso di successo, oppure **MBEDTLS_ERR_MD_BAD_INPUT_DATA** se i parametri non sono validi.

```
mbedtls_md_hmac_starts(&ctx, (const unsigned char *) key, keyLength);
```

Il passo successivo consiste nella chiamata alla funzione **mbedtls_md_hmac_update**, la quale riceve in ingresso il messaggio da autenticare (i parametri e il valore di ritorno sono analoghi a quelli della funzione vista in precedenza).

```
mbedtls_md_hmac_update(&ctx, (const unsigned char *) payload, payloadLength);
```

Ora che abbiamo specificato sia la chiave che il messaggio possiamo eseguire la chiamata alla funzione **mbedtls_md_hmac_finish** in modo tale da ottenere il codice di autenticazione prodotto. Il primo argomento è un puntatore al contesto, mentre il secondo è un buffer di byte per contenere il risultato dell'operazione. Il valore di ritorno si comporta esattamente come i casi precedenti.

```
mbedtls_md_hmac_finish(&ctx, hmacResult);
```

La successiva chiamata alla funzione **mbedtls_md_free** serve a liberare e azzerare le risorse precedentemente allocate (la funzione non restituisce alcun valore).

```
mbedtls_md_free(&ctx);
```

Il ciclo **for** successivo è auto esplicativo: il buffer contenente l'uscita dell'HMAC viene scandito byte a byte, convertito in esadecimale e inviato sulla linea seriale per una comoda visualizzazione.
Abbiamo a questo punto completato l'analisi del codice, il cui listato completo è consultabile alla pagina 107.

## TEST DELL'APPLICAZIONE

Per eseguire la verifica e la validazione del codice, è sufficiente caricare lo sketch in Arduino IDE, compilarlo e scaricarlo sulla board ESPertino.
Successivamente, con il monitor seriale aperto, lanciamo l'applicazione. Il trace prodotto sulla linea seriale sarà simile a quanto visibile in **Figura 5**.
In particolare, dovremo concentrarci sul **codice HMAC**, ovvero sulla sequenza di **32 byte** rappresentati in formato esadecimale. Per avere la conferma che il valore calcolato da ESPertino è corretto, possiamo utilizzare uno strumento disponibile online, in grado di calcolare l'HMAC di un messaggio fornito in ingresso applicando una tra N funzioni di hash disponibili.
Come visibile in **Figura 6** (nella barra degli URL è indicato l'indirizzo della pagina web), lo schermo di questo tool è suddiviso in tre parti: nella parte superiore andrà inserito il messaggio su cui calcolare l'HMAC, nella parte centrale andrà inserita la chiave segreta, mentre nella parte inferiore occorrerà selezionare l'algoritmo di hash da applicare (SHA256 nel nostro caso).
Premendo il tasto "COMPUTE HMAC" verrà calcolato e visualizzato il codice HMAC ottenuto dal tool applicando l'algoritmo SHA-256, come visibile in **Figura 7**.
A questo punto non ci resta che confrontare questa stringa di byte con quella prodotta dallo sketch eseguito su ESPertino: il risultato sarà identico!

## CONCLUSIONI

Abbiamo presentato in questo articolo un'utile applicazione crittografica di ESPertino.
Anche se non ce ne rendiamo sempre conto, l'acceleratore crittografico hardware integrato nel modulo ESP32 viene utilizzato in numerose applicazioni, come l'implementazione dei protocolli HTTPS e MQTT e in tutte le comunicazioni sicure basate su TLS/SSL.
La comunicazione sicura è un requisito fondamentale delle applicazioni IoT, dove le informazioni vanno preservate e protette da ogni possibile forma di attacco.
In questo contesto ESPertino si distingue per la completa dotazione di funzionalità hardware e librerie software in grado di supportare anche le più critiche applicazioni.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/applicazioni-crittografiche-di-espertino

# CRYPTO TICKER CON ESPERTINO: PROGETTO COMPLETO

di **Stefano Lovati**

*Grazie alle sue funzionalità avanzate, come la connettività e il supporto per numerosi tipi di interfacce, la scheda ESPertino si presta ad essere impiegata in applicazioni non solo utili ma anche insolite e divertenti. Il progetto presentato in questo articolo ha lo scopo di presentare su un display OLED le quotazioni in tempo reale di un certo numero di criptovalute, scelte a piacimento dall'utente.*

## INTRODUZIONE

Nonostante sia un sistema embedded di dimensioni molto compatte, **ESPertino** si dimostra un'ottima soluzione anche per sviluppare applicazioni nel campo delle **criptovalute**. In un precedente articolo, abbiamo visto come l'engine crittografico integrato nel modulo **ESP32** (il cuore della board) possa essere utilmente impiegato per il calcolo della funzione crittografica **SHA-256**, con numerose applicazioni nel campo della sicurezza (si pensi ad esempio ai protocolli sicuri e all'autenticazione). In questo articolo proponiamo un progetto differente che, oltre ad essere utile per chi vuole essere informato sulla quotazione corrente delle criptovalute, offre lo spunto per apprendere la tecnica di gestione di un **display OLED** con interfaccia **I²C**. Volete avere sempre sott'occhio la quotazione delle criptovalute preferite senza dover consultare continuamente i siti specializzati? Vi affascina il mondo delle monete digitali ma non sapete come entrare in questo ambiente in modo "soft", oppure volete semplicemente sperimentare la qualità superiore in termini di luminosità e contrasto di un display OLED? La risposta è molto semplice: vi basterà proseguire nella lettura dell'articolo! Il progetto ha lo scopo di realizzare un dispositivo che, collegandosi alla rete WiFi, visualizza su un display grafico il valore corrente di diverse criptovalute (è possibile gestire fino a un massimo di 10 differenti monete, selezionabili dall'utente). La quotazione delle criptovalute viene ottenuta accedendo al sito **CoinMarketCap.com** tramite apposite **API**, estraendo da esso le informazioni richieste. In pratica viene utilizzato il metodo GET del protocollo HTTP e la relativa risposta viene interpretata e decodificata utilizzando la libreria **ArduinoJson**. L'intero progetto è affrontabile da chiunque (non è richiesta alcuna saldatura o modifica hardware alla scheda): oltre a ESPertino è infatti sufficiente disporre di un display OLED (tra poco vedre-

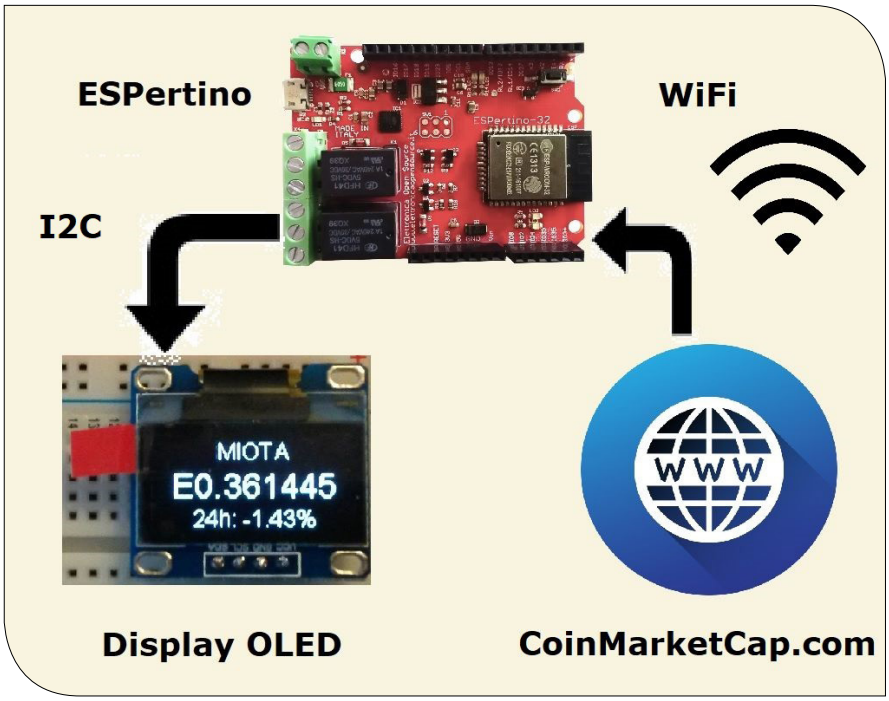



*Figura 1*

*Figura 2: uno dei più famosi esempi di stock ticker, installato a Times Square, New York*

mo i dettagli), una piccola protoboard e qualche cavetto jumper.
L'intera applicazione verrà sviluppata all'interno di **Arduino IDE**.

## SCHEMA A BLOCCHI

Lo schema a blocchi dell'intero progetto è visibile in **Figura 1**. La parte centrale del sistema è ovviamente rappresentata dalla scheda ESPertino, equipaggiata con il modulo ESP32 opportunamente programmato con lo sketch sviluppato in ambiente Arduino. Quest'ultimo, nella fase di setup, aprirà come client una connessione di rete collegandosi a un access provider (i corrispondenti parametri SSID e password andranno inseriti nello sketch, come vedremo). Una volta instaurata la connessione, l'applicazione comincerà a prelevare dal sito web CoinMarketCap le quotazioni correnti per le criptovalute configurate. Nello sketch è infatti presente una lista (di dimensione massima pari a 10) in cui l'utente può inserire le criptovalute di cui desidera monitorare la quotazione in tempo reale. Un set di valute è già preimpostato nello sketch, ma è possibile sostituire le criptovalute presenti con altre di proprio piacimento. La parte di acquisizione dei dati è visibile nella parte destra di **Figura 1**, schematizzata con il simbolo del world wide web e con l'indirizzo del sito che mette a disposizione le quotazioni. Si noti come la freccia sia entrante verso ESPertino, in quanto la scheda rappresenta il destinatario delle informazioni. In realtà, come sappiamo, il protocollo di comunicazione per l'accesso al sito (HTTP) implica sia la trasmissione che la ricezione dei dati.

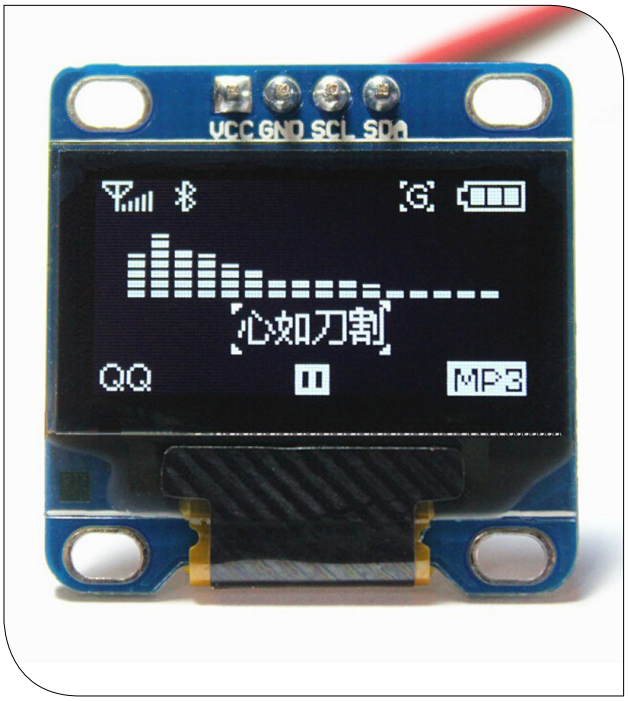



*Figura 3: display OLED basato sul controllore SSD1306*

Nella parte sinistra di **Figura 1** è invece presente il display, che riceve i dati da visualizzare (per comodità utilizzeremo una libreria apposita sviluppata per l'ambiente Arduino IDE, che si è dimostrata perfettamente compatibile anche con le schede basate sul microcontrollore ESP32). Il collegamento del display OLED con la scheda ESPertino avviene tramite interfaccia **$I^2C$**. Più

| Pin Display OLED | Pin ESPertino / serigrafia PCB |
|---|---|
| SDA | IO26 / x3 |
| SCL | IO27 |
| Vcc | 3V3 |
| GND | GND |

precisamente sono richiesti i seguenti **4 fili** (tutti i segnali sono disponibili sui connettori della board):

- 1 filo per il segnale serial clock (SCL);
- 1 filo per il segnale serial data (SDA);
- 1 filo per l'alimentazione positiva (+3,3 V);
- 1 filo per la massa (GND).

La libreria **Wire** (utilizzata per l'implementazione di basso livello del protocollo $I^2C$) abilita i pullup interni in corrispondenza dei segnali SCL e SDA e ciò, su molti tipi di display, potrebbe essere sufficiente al funzionamento corretto del dispositivo. È tuttavia preferibile inserire dei pullup esterni tra gli stessi segnali e Vcc, utilizzando resistenze con valore di 4,7 o 10 kΩ.

## ORIGINE DEL NOME TICKER

Qualche lettore potrebbe a questo punto chiedersi da dove derivi il nome assegnato al progetto. Nel mondo finanziario, per **stock ticker** si intende il prezzo associato a un determinato valore, aggiornato continuamente in seguito alle transazioni di acquisto e vendita eseguite sul medesimo (sia esso un titolo, una valuta, una criptovaluta o più in generale qualunque valore ammesso alle contrattazioni). In particolare, per "**tick**" si intende ogni variazione del prezzo corrente, che può essere sia al rialzo che al ribasso. Lo stock ticker ha quindi il compito di visualizzare automaticamente questi tick, insieme ad altre informazioni correlate (come i volumi scambiati nelle ultime 24 ore oppure nell'ultima settimana). Tali informazioni sono ovviamente utili agli investitori, ma anche a chi voglia rimanere informato sull'andamento di un titolo o valuta e monitorarne il valore nel tempo. Le informazioni visualizzate da uno stock ticker (nel nostro caso particolare si tratterà di un **crypto ticker**) sono aggiornate in tempo reale oppure con un leggerissimo ritardo. Uno degli stock ticker più famoso è sicuramente quello visibile a New York in Times Square. Come visibile in **Figura 2**, un sistema completamente elettronico visualizza l'andamento dei titoli tecnologici quotati al Nasdaq (le barre verdi indicano un rialzo, quelle rosse un ribasso del titolo).

## DISPLAY OLED

Il display scelto per l'applicazione è basato sul diffuso componente **SSD1306**, un driver monolitico per la gestione di display grafici OLED/PLED. Il controllore gestisce una matrice di OLED composta da **128 segmenti e 64 linee comuni** e pertanto la risoluzione del display è pari a 128 x 64 pixel. Il componente SSD1306 include un circuito per il controllo automatico del contrasto, la memoria RAM per il video e un oscillatore, minimizzando gli assorbimenti e il numero di componenti esterni richiesto. La versione di display utilizzata per il progetto è controllabile dal sistema host (il microcontrollore ESP32 che equipaggia ESPertino) tramite l'invio di dati e comandi su interfaccia $I^2C$. Il display (visibile in **Figura 3**) ha dimensioni molto compatte, con una diagonale pari a **0,96 pollici**. Nonostante questo, lo schermo è perfettamente leggibile anche nelle condizioni più critiche di luminosità e di incidenza del fascio luminoso: ciò avviene in virtù dell'elevato contrasto e nitidezza tipici dei display OLED. Inoltre, poiché la radiazione luminosa è emessa dagli stessi led organici, non è richiesto alcun sistema di retroilluminazione, con conseguenti vantaggi in termini di riduzione dei consumi. Un ulteriore significativo vantaggio è offerto dalla presenza, integrata sul PCB del display, di appositi regolatori di tensione che ne permettono il funzionamento con valori di alimentazione compresi tra 3V e 5V. È così possibile utilizzare il medesimo display per progetti basati ad esempio su una scheda Arduino Uno oppure su una scheda ESPertino, senza la necessità di introdurre dei traslatori di livello.
Pur essendo monocromatico, esistono in commercio versioni del display con colorazione dei led bianchi oppure blu. Entrambe le versioni sono adatte al progetto, la cosa importante è verificare che l'interfaccia utilizzata sia $I^2C$ (la serigrafia del PCB deve mostrare i segnali come indicato in **Figura 3**: SDA, SCL, Vcc e GND). Per quanto concerne il collegamento del display alla scheda ESPertino, si può fare riferimento alle indicazioni fornite nella **Tabella 1**.
Per comodità, gli stessi collegamenti sono proposti in **Figura 4** (il layout della scheda è stato tratto dal data-

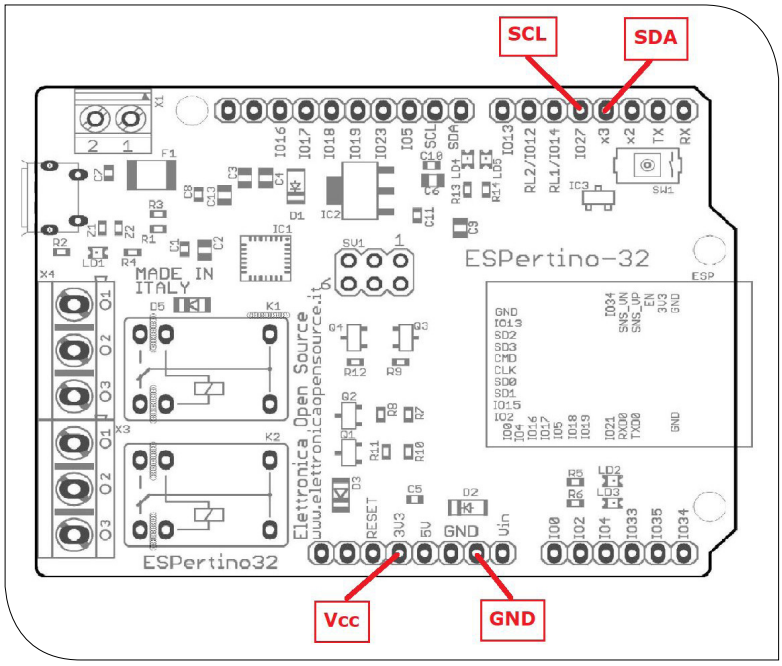



*Figura 4: evidenziazione dei pin e connettori di ESPertino utilizzati per il collegamento con il display*

*Figura 5: layout dei dati visualizzati sul display OLED*

sheet di ESPertino).

In **Figura 5** possiamo invece osservare il layout dei dati visualizzati sul display. La prima riga mostra il nome abbreviato della criptovaluta, un codice univoco valido universalmente che è ad esempio visibile nella home page del sito CoinMarketCap (colonna "Circulating Supply"). Nel caso di **Figura 5**, ad esempio è visualizzata la quotazione della moneta Dogecoin (DOGE). La seconda riga (che utilizza un font di dimensioni maggiori) mostra invece la quotazione della criptovaluta fornita dallo stesso sito web. Tale valore è preceduto dalla lettera 'E' per indicare che il valore è riferito alla valuta euro. La terza ed ultima riga riporta infine la variazione percentuale della criptovaluta registrata nelle ultime 24 ore. I dati visualizzati sono continuamente aggiornati scorrendo la lista di criptovalute desiderate codificata nello sketch (attualmente la lista può essere composta al massimo da 10 monete digitali differenti).

## API COINMARKETCAP

Per estrarre i dati relativi alla quotazione delle criptovalute è stata utilizzata l'interfaccia per gli sviluppatori (**API**) messa a disposizione dal sito **CoinMarketCap**. Questo sito gode di una certa fama sia per l'ampia disponibilità di dati, sia per il fatto di avere rilasciato un'interfaccia pubblica (le versioni V1 e V2, leggermente differenti per quanto riguarda la struttura e il contenuto dei dati esposti sono documentate in [1] e [2]). A partire dal 4 dicembre 2018, tuttavia, queste due modalità di accesso ai dati sono state sostituite da una nuova versione di interfaccia: la **Professional API**. La principale differenza è che ora occorre utilizzare una **chiave privata di accesso** (rilasciata dal sito previa registrazione) e inserirla opportunamente nella richiesta HTTP eseguita per ottenere i dati. Esistono diversi profili utente, alcuni a pagamento con funzionalità e servizi sempre più avanzati e completi. Per la nostra applicazione utilizzeremo comunque la **modalità free**, che non richiede la stipula di alcun contratto o pagamento: è sufficiente registrarsi al sito fornendo il proprio nome e indirizzo mail, scegliere la modalità "Basic Plan" e successivamente ottenere la chiave da utilizzare con le API. si osservi a questo proposito la **Figura 6**, che mostra la dashboard una volta eseguito il login. Cliccando sul tasto "Copy key" a sinistra è possibile copiare il contenuto della chiave privata nello scratchpad (la chiave andrà inserita nello sketch, come vedremo tra poco). Si noti inoltre come la dashboard riporti il **numero di crediti** (credits) utilizzati nel giorno corrente e nell'ultimo mese. I crediti sono limitati nella versione free (333 al giorno), per cui occorre ge-

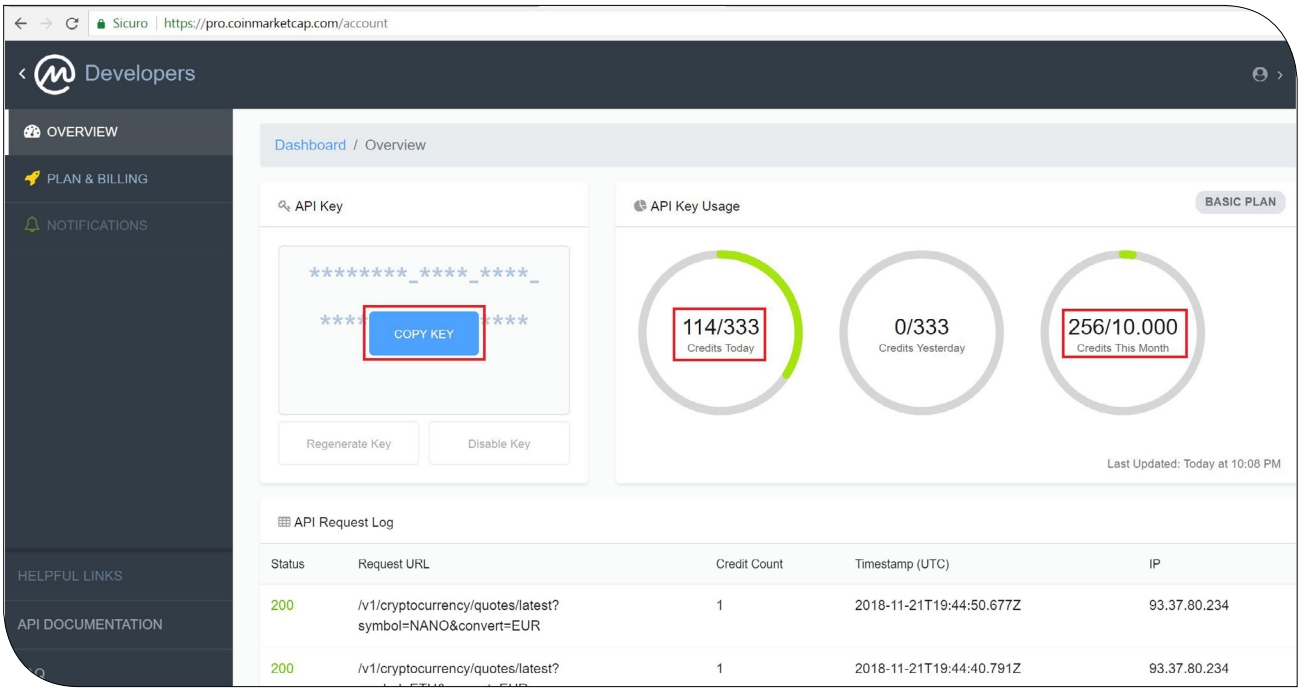



*Figura 6: il pannello di controllo utente da cui può essere recuperata la chiave privata e il residuo dei crediti*

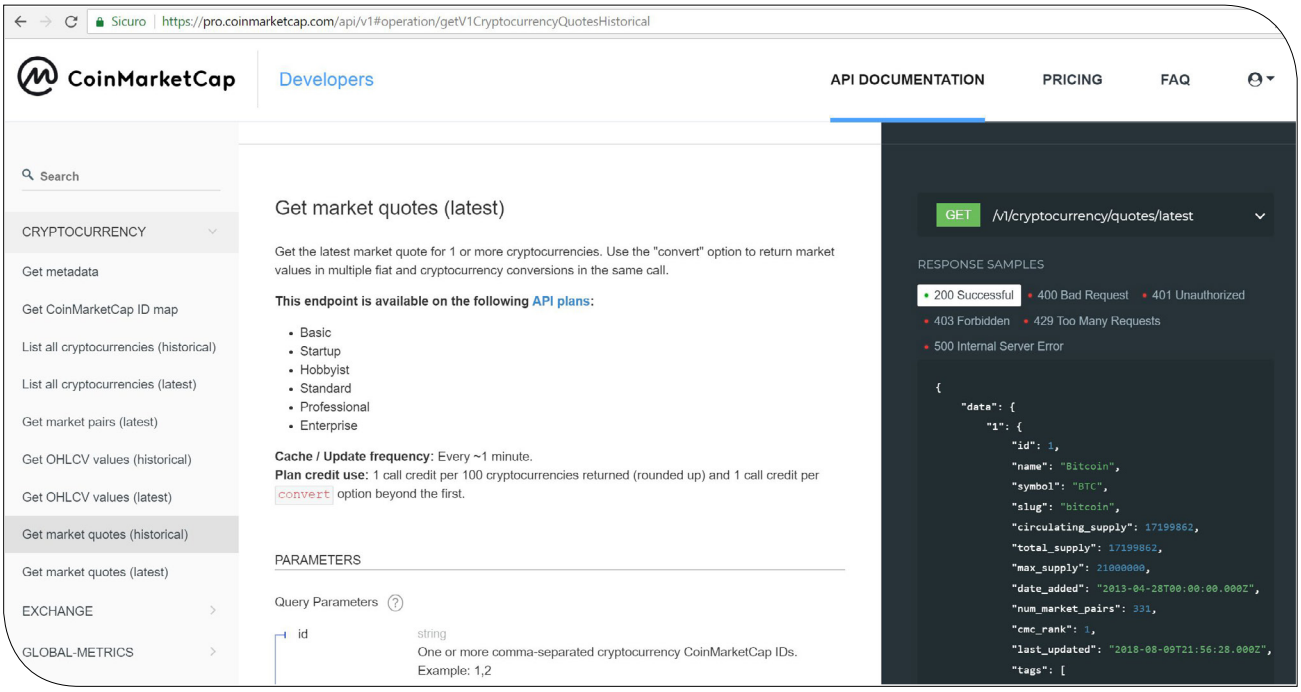



*Figura 7: la documentazione delle API è completa e molto dettagliata*

stirli opportunamente. Nel caso di interrogazione di 10 criptovalute (inclusa la conversione automatica in euro eseguita dal servizio), vengono consumati 10 crediti ogni volta.

Per quanto riguarda il formato dei dati, l'interfaccia è di tipo **JSON**, gestita nell'applicazione tramite la libreria **ArduJson**. La documentazione ufficiale delle API [3] è molto completa e facilmente comprensibile, anche se in lingua inglese. Più precisamente, il comando utilizzato all'interno dello sketch è il "Get market quotes (latest)" (in **Figura 7** è visualizzata la pagina della relativa documentazione), che restituisce la quotazione corrente (ultima in ordine di tempo) della criptovaluta specificata come parametro del metodo HTTP GET. A destra in **Figura 7** è visibile un esempio di Json restituito dalle API (in questo caso si tratta di una richiesta relativa alla

moneta “BTC”, cioè il Bitcoin). Questa parte è completamente implementata nello sketch, che è quindi compatibile alla versione Professional delle API e non richiede alcuna modifica se non l’inserimento della propria chiave privata ottenuta previa registrazione di un account free su CoinMarketCap.
A titolo di esempio, riportiamo il Json completo restituito dall’API precedentemente citata a seguito di una richiesta relativa alla criptovaluta Monero (“XMR”):


```
{
  “status”: {
    “timestamp”: “2018-11-21T21:27:43.812Z”,
    “error_code”: 0,
    “error_message”: null,
    “elapsed”: 6,
    “credit_count”: 1
  },
  “data”: {
    “XMR”: {
      “id”: 328,
      “name”: “Monero”,
      “symbol”: “XMR”,
      “slug”: “monero”,
      “circulating_supply”: 16587767.6143826,
      “total_supply”: 16587767.6143826,
      “max_supply”: null,
      “date_added”: “2014-05-21T00:00:00.000Z”,
      “num_market_pairs”: 105,
      “tags”: [
        “mineable”
      ],
      “cmc_rank”: 10,
      “last_updated”: “2018-11-21T21:27:08.000Z”,
      “quote”: {
        “EUR”: {
          “price”: 60.042084403353385,
          “volume_24h”: 18859128.49430606,
          “percent_change_1h”: -0.2479,
          “percent_change_24h”: 3.4633,
          “percent_change_7d”: -33.5277,
          “market_cap”: 995964143.1659719,
          “last_updated”: “2018-11-21T21:25:00.000Z”
        }
      }
    }
  }
}
```


## LIBRERIE RICHIESTE

L’applicazione software si basa sulle seguenti componenti:

- sketch Arduino, di cui parleremo nel prossimo paragrafo;
- la classe **CoinMarketCapProfessionalApi**, definita nei file CoinMarketCapProfessionalApi.h e CoinMarketCapProfessionalApi.cpp, presenti nella stessa cartella dello sketch. Tale classe definisce la struttura dati e i metodi per l’interfacciamento con le API Professional esposte da CoinMarketCap;
- la libreria **ArduinoJson**, necessaria per eseguire in modo agevole il parsing della struttura Json restituita a seguito dell’interrogazione HTTP GET;
- la **libreria** per pilotare il **display** OLED.

Entrambe le librerie vanno installate nell’IDE Arduino attraverso la voce del menu: Sketch > Include Library > Manage Libraries. Nel primo caso, occorre inserire nella barra di ricerca “ArduinoJson”.
Nei risultati della ricerca selezionare e installare la libreria **ArduinoJson** di Benoit Blanchon, **versione 5.13.2** (evitare assolutamente la versione beta, non è adatta alla nostra applicazione), come indicato in **Figura 8**. Per quanto riguarda invece la libreria per il display OLED, occorre procedere esattamente nello stesso modo, inserendo questa volta nella barra di ricerca la stringa “oled driver”. Nei risultati della ricerca, selezionare e installare la libreria “ESP8266 and ESP32 Oled Driver for SSD1306 display” di Daniel Eichhorn e Fabrice Weinberg, versione 4.0.0 (si osservi la **Figura 9**).

## LO SKETCH

Lo sketch è contenuto nel file ESPertino_Crypto_ticker.ino, allegato all’articolo insieme ai due file che definiscono la classe CoinMarketCapProfessionalApi. Prima di eseguire la compilazione e il download dello sketch su ESPertino occorre comunque modificare a mano la seguente sezione, contenente le **credenziali di accesso alla rete WiFi** e la **chiave personale** per accedere alle API Professional di CoinMarketCap:

```
// Configuratione della rete WiFi e della chiave
privata
// per l’accesso alle API Professional di CoinMar-
ketCap
//
// inserire qui i propri valori prima di compilare lo
 sketch!!!
```

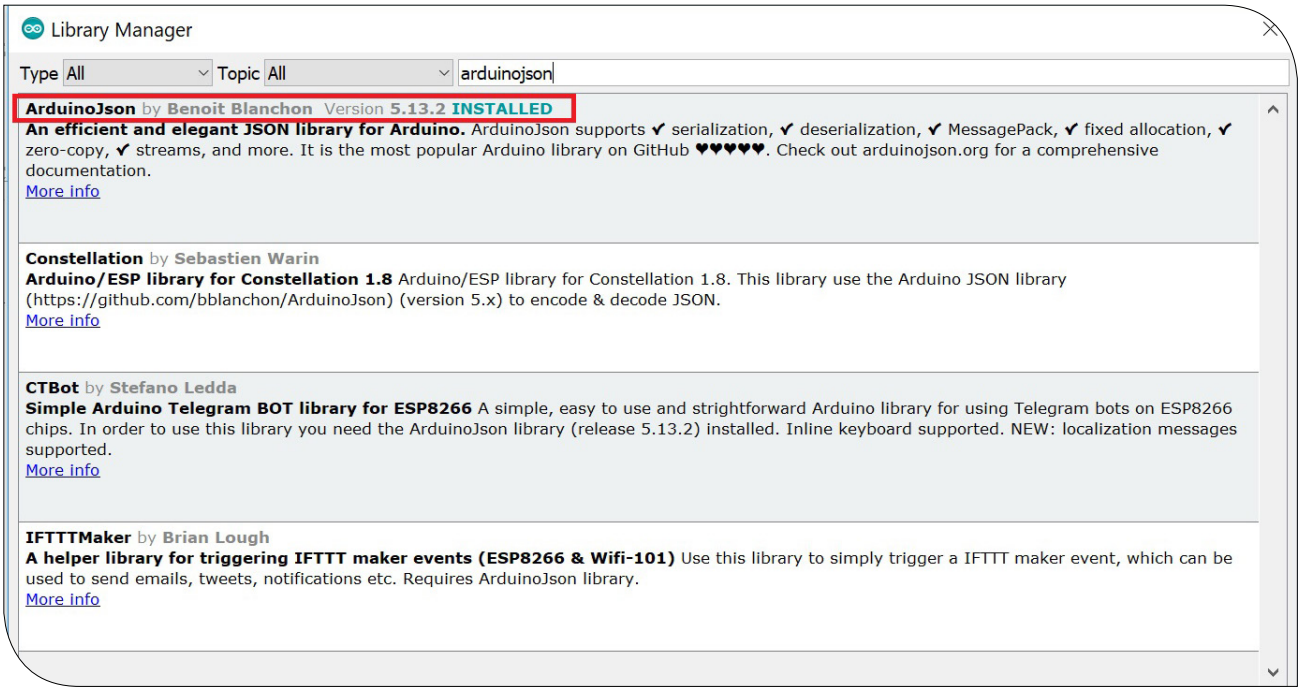



*Figura 8: selezione e installazione della libreria ArduinoJson ver. 5.13.2*

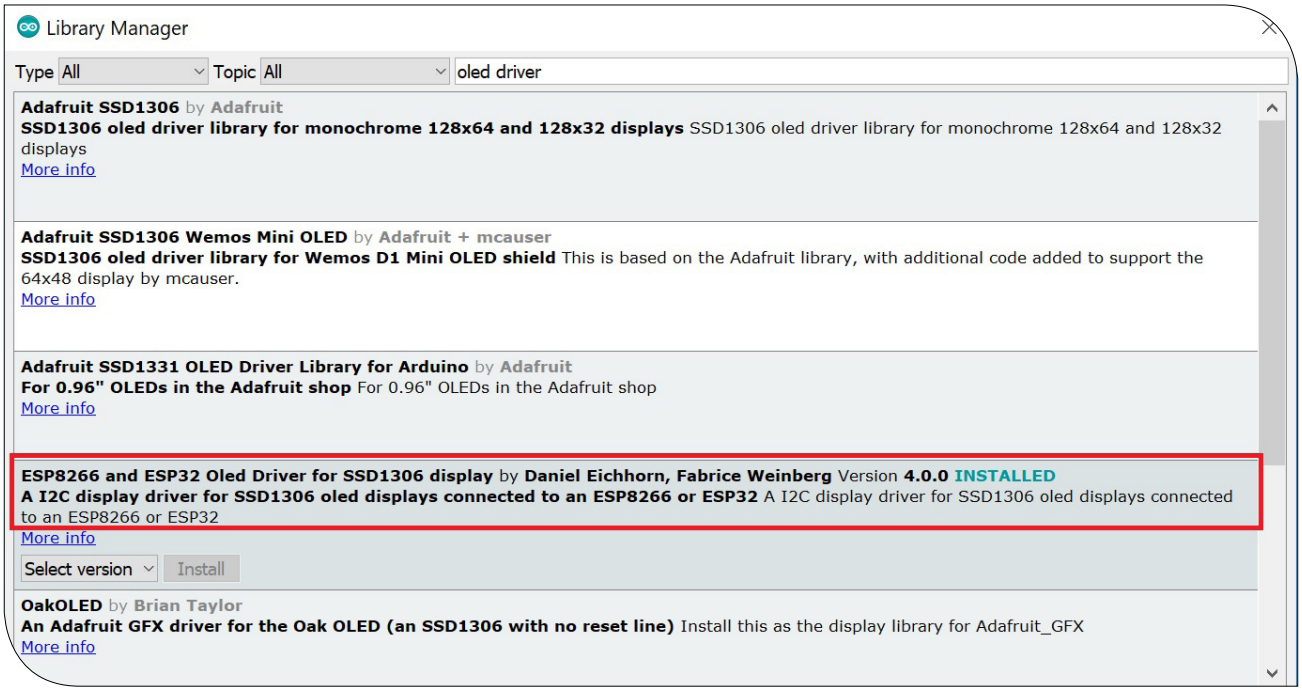



*Figura 9: selezione e installazione della libreria per il display OLED*

```
//
char glb_ssid[] = "XXXXXXXXXX";
char glb_password[] = "XXXXXXXXXX";
char glb_api_key[] = "X-CMC_PRO_API_KEY:
XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX";
```

Al primo avvio il programma visualizzerà una schermata di benvenuto (visibile in **Figura 10**), per poi cominciare a proporre la quotazione corrente e la variazione percentuale sulle ultime 24 ore delle criptovalute inserite nell'apposita lista dello sketch.
La lista delle criptovalute gestite, contenuta all'interno

dello sketch, è la seguente:

```
cripto_info_t glb_cripto_info[MAX_NUM_CRIPTOVALUTE] = {
 { "Bitcoin",  "BTC",   0.0F, 0.0F, true, PRESCALER_TICKER_GET },
 { "Dogecoin", "DOGE",  0.0F, 0.0F, true, PRESCALER_TICKER_GET },
 { "Ethereum", "ETH",   0.0F, 0.0F, true, PRESCALER_TICKER_GET },
 { "Nano",     "NANO",  0.0F, 0.0F, true, PRESCALER_TICKER_GET },
 { "Digibyte", "DGB",   0.0F, 0.0F, true, PRESCALER_TICKER_GET },
 { "Iota",     "MIOTA", 0.0F, 0.0F, true, PRESCALER_TICKER_GET },
 { "Monero",   "XMR",   0.0F, 0.0F, true, PRESCALER_TICKER_GET },
 { "MonaCoin", "MONA",  0.0F, 0.0F, true, PRESCALER_TICKER_GET },
 { "Bytecoin", "BCN",   0.0F, 0.0F, true, PRESCALER_TICKER_GET },
 { "Litecoin", "LTC",   0.0F, 0.0F, true, PRESCALER_TICKER_GET }
};
```

*Figura 10: schermata di benvenuto dell'applicazione*

Il primo campo è il nome della criptovaluta: serve solo come memo, non è visualizzato sul display e non è utilizzato dallo sketch. Il secondo campo è il simbolo univoco della moneta digitale, mentre il penultimo campo è la flag di abilitazione: se true il valore di quella criptovaluta viene acquisito e visualizzato, se false la criptovaluta viene ignorata. Risulta così possibile ridurre, se lo si desidera, il numero di valute visualizzate sul display. I valori relativi a ogni criptovaluta rimangono visualizzati sul display per 10 secondi; con 10 valute il ciclo completo di visualizzazione dura pertanto 100 secondi. Poiché esiste un limite di 330 crediti al giorno (nel nostro caso corrispondono esattamente a 330 richieste eseguite tramite le API Professional) è stato introdotto un meccanismo regolato dal valore della seguente costante:

```
#define PRESCALER_TICKER_GET 27 // periodo di acquisizione ticker = 45 minuti
```

Ad ogni ciclo completo di visualizzazione (100 secondi nel caso peggiore) viene decrementata una variabile inizializzata al valore di tale costante. L'acquisizione dei dati tramite API viene eseguita una volta ogni PRESCALER_TICKER_GET cicli completi, preservando così il numero disponibile di crediti per l'intera giornata. Tra una acquisizione e la successiva vengono visualizzati gli ultimi valori ottenuti da CoinMarketCap.

## CONCLUSIONI

Il progetto presentato nell'articolo rappresenta non soltanto una utile occasione per apprendere concetti e strumenti comunemente utilizzati nel mondo delle criptovalute, ma è anche un modo divertente e pratico per imparare a utilizzare nuove periferiche per ESPertino (come il display OLED con interfaccia $I^2C$) e nuove metodologie di acquisizione dei dati (interfaccia Json esposta dal sito CoinMarketCap).

## LINK ALLO SKETCH

*Sketch per il progetto ESPertino Crypto Ticker*

## RIFERIMENTI

[1] https://coinmarketcap.com/it/api/documentation/v1/
[2] https://coinmarketcap.com/api/
[3] https://pro.coinmarketcap.com/api/v1

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/crypto-ticker-con-espertino-progetto-completo

# COME INTEGRARE DISPOSITIVI IOT NELLA RETE IOTA

di **Stefano Lovati**

*Scopriamo in questo articolo, dal taglio prevalentemente applicativo (o pratico), come sia possibile collegare alla rete IOTA vari tipi di dispositivi elettronici, realizzando e sperimentando l'utilizzo della criptovaluta in un sistema IoT reale. Con un Raspberry Pi e pochi altri componenti dimostreremo infatti come sia possibile eseguire dei pagamenti e ottenere dei servizi da un dispositivo fisico*

## INTRODUZIONE

**IOTA** è una criptovaluta a cui abbiamo dedicato uno spazio considerevole, con una serie di articoli tematici di presentazione e di approfondimento delle sue funzionalità. Il motivo principale di questa scelta risiede nell'applicazione tipica a cui è destinata IOTA, cioè il mondo delle applicazioni **IoT**. Grazie a caratteristiche quali scalabilità, efficienza, rapidità nella creazione delle transazioni e all'utilizzo della struttura **Tangle** (un particolare tipo di grafo aciclico diretto, o DAG), IOTA è in grado di supportare agevolmente le transazioni tipiche di un sistema IoT, caratterizzato da micro pagamenti che devono necessariamente essere completati in tempi brevi e in modo totalmente sicuro. In questo articolo, una sorta di guida rivolta a tutte le tipologie di utenti (compresi i principianti) vedremo come seguendo pochi e semplici passi sia possibile utilizzare il protocollo su cui si basa IOTA per effettuare dei micro pagamenti e per ricevere, in cambio, i servizi offerti da un dispositivo fisico reale. Più precisamente, l'obiettivo del progetto presentato nell'articolo è quello di realizzare un semplice circuito che possa essere attivato (alimentato) oppure disattivato basandosi sulla disponibilità di fondi in un determinato wallet/indirizzo IOTA. In modo analogo a molte altre criptovalute, IOTA permette infatti di creare un wallet (portafoglio) al quale viene associato un indirizzo univoco; tramite opportune applicazioni software disponibili per tutte le piattaforme hardware e per i principali tipi di sistema operativi, è poi possibile eseguire delle transazioni di pagamento (prelevando l'importo in IOTA dal wallet selezionato) oppure ricevere dei pagamenti eseguiti in IOTA (fornendo l'indirizzo del proprio wallet come destinatario della transazione). L'idea di **associare a una transazione di pagamento l'attivazione di una funzionalità hardware o software** o in genere il controllo di un'apparecchiatura elettronica non è poi così lontana dalla realtà: anzi, IOTA è nata soprattutto per supportare le transazioni **M2M** tipiche del mondo IoT, dove occorre operare con efficienza e sicurezza. Le transazioni possono anche avere importo molto ridotto (si parla, appunto, di micro pagamenti) e non vi è la necessità di "attirare" i miner con commissioni generose poiché in IOTA non esistono miner: la transazione verrà validata e inserita nel Tangle senza dover sborsare alcuna commissione extra. Qualche sviluppatore o ricercatore ha ipotizzato scenari simili a quelli prospettati ma molto più pratici. È

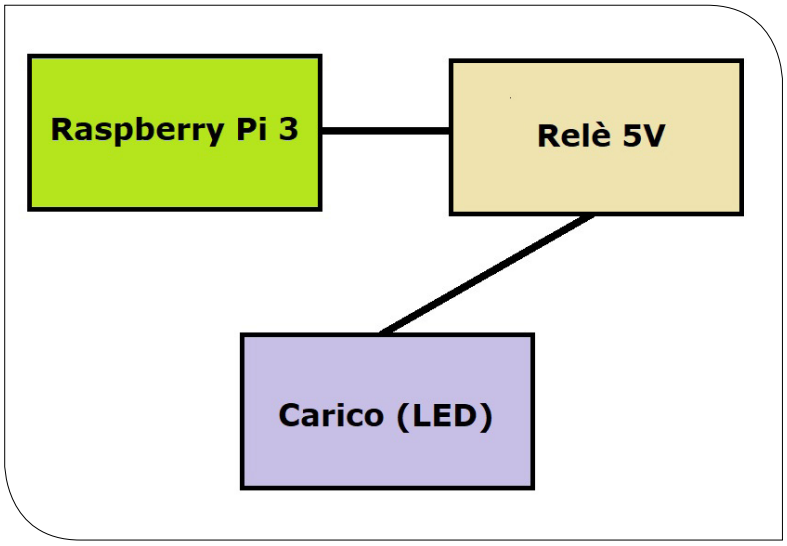



*Figura 1: schema a blocchi dell'applicazione*

Figura 2: Raspberry Pi 3 B/B+: il cuore del progetto

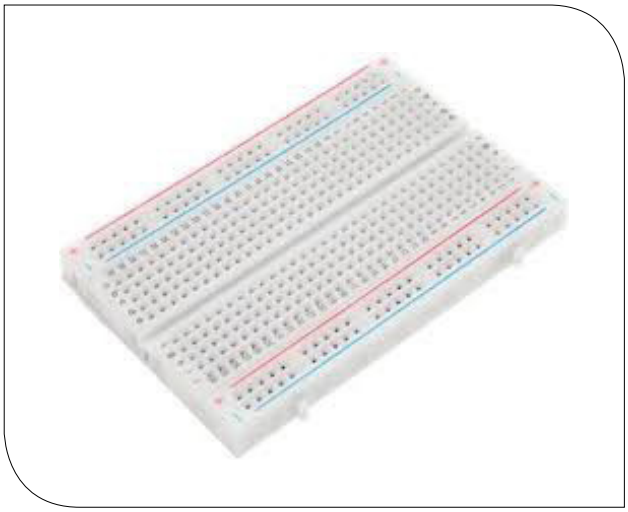


Figura 4: la breadboard su cui assemblare il circuito

Figura 3: il modulo relè

Figura 5: il diodo LED è di tipo normale, per uso convenzionale

infatti possibile trovare in rete più di un video che mostra come attivare una macchinetta del caffè tramite il pagamento eseguito con una criptovaluta: chissà, forse in un prossimo futuro vedremo applicazioni di questo tipo nella vita di tutti i giorni, a partire dalla colazione al bar.

## IL PROGETTO

Il cuore del progetto è rappresentato dalla scheda **Raspberry Pi 3 B/B+** capace di connessione a internet tramite l'interfaccia WiFi integrata. Con riferimento allo schema a blocchi di **Figura 1**, oltre al Raspberry Pi il progetto include i seguenti componenti:

- modulo relè compatibile con il livello di tensione presente sui GPIO del Raspberry Pi (3,3 V);
- una piccola scheda di prototipazione rapida (breadboard);
- un LED di qualsiasi tipo per impieghi convenzionali;
- una pila da 9 V;
- una resistenza R da 330 Ω;
- cavetti jumper.

Il principio di funzionamento del circuito è molto semplice. Il Raspberry Pi, opportunamente programmato con un'applicazione scritta in **Python**, verifica periodicamente il saldo del conto in IOTA accedendo direttamente alla rete Tangle. Se il saldo è positivo, viene attivato un GPIO che a sua volta pilota un relè. All'uscita del relè è collegato un circuito composto dalla pila, dal LED e dalla resistenza R. Il LED dovrà accendersi solo se il saldo associato all'indirizzo IOTA è positivo, in caso contrario rimarrà spento.

Qualcuno potrebbe a questo punto chiedersi perché non pilotare direttamente il LED tramite un GPIO in uscita dal Raspberry Pi. La domanda è lecita, ma un motivo per cui questa scelta è stata effettuata esiste. Anziché optare per la versione "semplificata" del circuito (priva del relè) si è deciso di proporre una versione che permetta (a discrezione dell'utilizzatore finale) di pilotare diverse tipologie di carichi in uscita, non soltanto un semplice diodo LED. Risulta così possibile collegare dispositivi o carichi con potenza superiore (compatibilmente con il tipo di relè impiegato) e si ottiene nello

*Figura 6: resistenza di limitazione per il diodo LED*

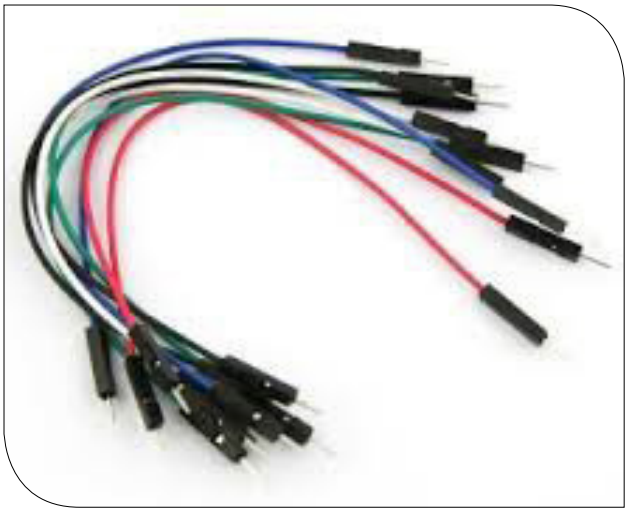


*Figura 8: cavetti jumper richiesti per il cablaggio*

*Figura 7: una comune pila a 9V*

*Figura 9: QR code per semplificare l'operazione di pagamento*

stesso tempo un buon grado di isolamento tra la scheda Raspberry Pi e il circuito pilotato.

## PRINCIPIO DI FUNZIONAMENTO

Come detto in precedenza, anche se relativamente semplice, questo progetto può trovare delle applicazioni pratiche nella vita di tutti i giorni, aiutandoci a risolvere dei problemi reali. Supponiamo ad esempio di soggiornare in un hotel in cui ogni stanza sia dotata di un proprio frigorifero. Nella maggior parte dei casi, questi frigoriferi sono sempre accesi, consumando energia indipendentemente dal loro reale utilizzo e contribuendo indirettamente al costo della stanza. Perché allora non pensare a un meccanismo che consenta di pagare il frigorifero solo per il tempo in cui è effettivamente utilizzato, spegnendolo automaticamente quando non serve più? Fondamentalmente, questo è proprio il caso di utilizzo (detto anche use case) che andremo a realizzare in questo articolo, con la differenza che (per ragioni di economicità, praticità e comodità) il frigorifero sarà sostituito da un comune LED. Immaginiamo ora una sequenza di eventi che dimostri come il sistema possa essere implementato e utilizzato. Supponiamo anzitutto che il proprietario dell'hotel abbia installato un frigorifero nella nostra stanza, collocando un relè nel circuito di alimentazione dell'elettrodomestico. Il relè è a sua volta collegato a un GPIO di una scheda Raspberry Pi, utilizzata in veste di unità di controllo per il sistema di pagamento. Successivamente, il proprietario dell'hotel crea un indirizzo IOTA associato al frigorifero per monitorare quando vengono ricevuti nuovi fondi. Infine, egli crea e stampa un codice QR associato all'indirizzo IOTA e lo attacca al telaio del frigorifero. La configurazione hardware è ora completa: il proprietario dell'hotel non dovrà fare altro che programmare il Raspberry Pi con un'applicazione Python che controlla l'accredito di nuovi fondi sull'indirizzo IOTA, accendendo o spegnendo il frigorifero conseguentemente. Supponiamo ora di essere nei panni di un cliente dell'hotel, che ha appena acquistato una bottiglia di ottimo vino bianco, da gustare più tardi durante la serata. Per essere certi che il vino sia servito alla temperatura ottimale, il cliente prende il proprio

smartphone, apre l'applicazione per wallet IOTA su esso installata, scansiona il codice QR presente sul frigorifero ed esegue il pagamento in IOTA per l'utilizzo dell'elettrodomestico (possiamo supporre che l'entità della somma da trasferire sia funzione del tempo di utilizzo del servizio). Non appena la transazione viene confermata dal Tangle, il saldo del frigorifero viene incrementato, provocando come conseguenza diretta l'attivazione del relè e la conseguente accensione del frigorifero. Ecco quindi un modo per gustarsi del buon vino alla giusta temperatura, nel modo più moderno, efficiente e sicuro possibile.

## COMPONENTI

Vediamo ora in dettaglio i componenti richiesti per la realizzazione del progetto, tutti di facile reperibilità e di cui molti lettori saranno probabilmente già in possesso. Il cuore del progetto è rappresentato dalla scheda **Raspberry Pi** (visibile in **Figura 2**), il cui compito è quello di monitorare sul Tangle l'indirizzo IOTA e in caso di nuovi fondi gestire opportunamente lo stato del relè collegato ai GPIO della scheda.

Abbiamo poi il **modulo relè**, utilizzato per attivare e disattivare il carico, rappresentato nel nostro caso da un semplice LED). Abbiamo deciso di utilizzare un modulo (anziché un relè normale) in quanto il primo è già corredato di tutti i componenti e connettori necessari per eseguire un cablaggio agevole. Moduli di questo tipo (un esempio è visibile in **Figura 3**) sono molto economici e largamente utilizzati anche sulla piattaforma Arduino. Se già lo si possiede, si può anche utilizzare un modulo con relè (canali) multipli, indirizzabili singolarmente. Ciò può essere utile nei casi in cui si vogliano gestire più dispositivi (carichi) in uscita.

La piccola **breadboard**, visibile in **Figura 4**, ha il compito di semplificare il cablaggio del prototipo, non richiedendo alcun tipo di saldatura. Risulta inoltre molto semplice l'operazione di smantellamento del circuito, con i componenti ancora integri che possono essere nuovamente riutilizzati.

Il diodo **LED** (**Figura 5**) non ha particolari requisiti, si può benissimo utilizzare un comune LED per impieghi convenzionali.

La **resistenza** (visibile in **Figura 6**) ha il compito di limitare la corrente che attraversa il diodo LED.

Utilizzando una pila da 9V, dovrebbe bastare un valore di resistenza da 330 Ω o 470 Ω (si possono utilizzare anche valori superiori, ma ovviamente la luminosità del diodo LED risulterà inferiore).

La **pila da 9V**, visibile in **Figura 7**, ha il compito di fornire l'alimentazione al carico, mentre i **cavetti jumper** (un esempio è mostrato in **Figura 8**) sono necessari per effettuare i cablaggi che non richiedono alcuna saldatura. Si consiglia di utilizzare cavetti jumper di vario tipo (maschio-maschio e femmina-femmina) in modo tale da coprire tutte le possibili casistiche di collegamento.

Il **codice QR** associato al pagamento IOTA (visibile in **Figura 9**) è comodo se si vuole eseguire il pagamento utilizzando un'applicazione per wallet IOTA su dispositivo mobile.

Tali codici QR possono essere ottenuti quando si generano nuovi indirizzi utilizzando un wallet IOTA oppure eseguendo la ricerca di un indirizzo esistente su un apposito sito web [1].

## CABLAGGIO

Tutte le informazioni richieste per portare a termine con successo il cablaggio sono riassunte nello schema Fritzing di **Figura 10**, mentre in **Figura 11** è riproposto (per comodità del lettore) il connettore con indicati i GPIO del Raspberry Pi.

I collegamenti richiesti per portare a termine il cablaggio sono i seguenti:

- il pin 2 (5V) del Raspberry Pi va collegato al pin Vcc del modulo relè;
- il pin 6 (GND) del Raspberry Pi va collegato al pin GND del modulo relè;
- il pin 12 (GPIO18) del Raspberry Pi va collegato al pin IN del modulo relè;
- il terminale COM del modulo relè va collegato al polo positivo della pila;
- il terminale NO del modulo relè va collegato al terminale positivo (anodo) del LED, con la resistenza inserita tra i due;
- il polo negativo della pila va collegato al terminale negativo (catodo) del LED.

## SOFTWARE E LIBRERIE

Anche se in linea di principio qualunque distribuzione basata su Linux va bene per i nostri scopi, è consigliabile procedere anzitutto con l'installazione della **distribuzione Raspbian**, scaricabile direttamente dal sito ufficiale della Raspberry Pi Foundation [2]. Oltre ad essere largamente utilizzata dalla comunità di sviluppatori e utenti, Raspbian ha il vantaggio di avere **Python** già installato, oltre a diversi editor e IDE specifici per questo linguaggio. Come supporto di memoria è consigliabile disporre di una **scheda SD** con dimensione di almeno 8 Gb, preferibilmente di buona qualità e ad alta velocità. L'immagine di Raspbian può essere programmata sulla

```
# Imports some Python Date/Time functions
import time
import datetime

# Imports GPIO library
import RPi.GPIO as GPIO

# Imports the PyOTA library
from iota import Iota
from iota import Address

# Setup O/I PIN's
LEDPIN=18
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(LEDPIN,GPIO.OUT)
GPIO.output(LEDPIN,GPIO.LOW)

# Function for checking address balance on the IOTA tangle.
def checkbalance():
    print("Checking balance")
    gb_result = api.get_balances(address)
    balance = gb_result['balances']
    return (balance[0])

# URL to IOTA fullnode used when checking balance
iotaNode = "https://field.deviota.com:443"

# Create an IOTA object
api = Iota(iotaNode, "")

# IOTA address to be checked for new light funds
# IOTA addresses can be created using the IOTA Wallet
address = [Address(b'GTZUHQSPRAQCTSQBZEEMLZPQUPAA9LPLGWCKFNEVKBINXEXZRAC-
VKKKCYPWPKH9AWLGJHPLOZZOYTALAWOVSIJIYVZ')]

# Get current address balance at startup and use as baseline for measuring new funds being added.
currentbalance = checkbalance()
lastbalance = currentbalance

# Define some variables
lightbalance = 0
balcheckcount = 0
lightstatus = False

# Main loop that executes every 1 second
while True:
    # Check for new funds and add to lightbalance when found.
    if balcheckcount == 10:
        currentbalance = checkbalance()
        if currentbalance > lastbalance:
            lightbalance = lightbalance + (currentbalance - lastbalance)
            lastbalance = currentbalance
        balcheckcount = 0

    # Manage light balance and light ON/OFF
    if lightbalance > 0:
        if lightstatus == False:
            print("light ON")
            GPIO.output(LEDPIN,GPIO.HIGH)
            lightstatus=True
        lightbalance = lightbalance -1
    else:
        if lightstatus == True:
            print("light OFF")
            GPIO.output(LEDPIN,GPIO.LOW)
            lightstatus=False

    # Print remaining light balance
    print(datetime.timedelta(seconds=lightbalance))

    # Increase balance check counter
    balcheckcount = balcheckcount +1

    # Pause for 1 sec.
    time.sleep(1)
```

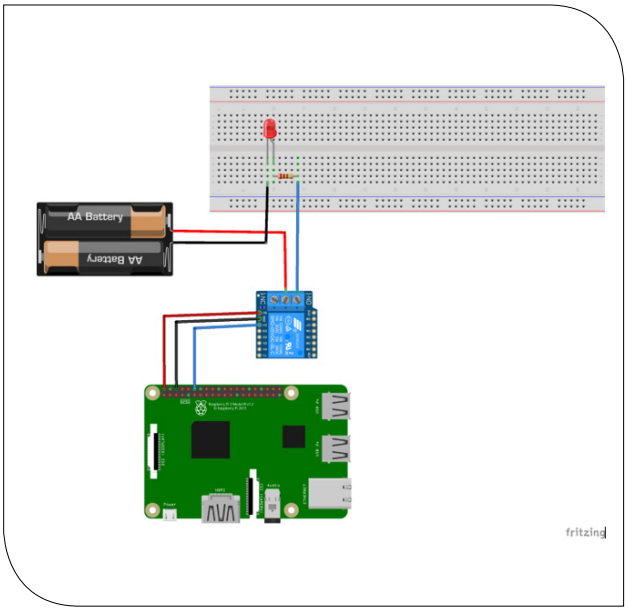



*Figura 10: schema Fritzing con il cablaggio completo del circuito*

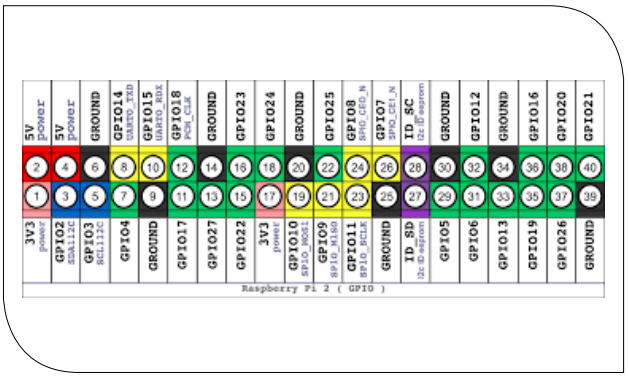



*Figura 11: indicazione dei GPIO presenti sul connettore del Raspberry Pi 3*

scheda SD tramite **Etcher** [3] o strumenti simili. Alla prima accensione con Raspbian, configurare la lingua, la risoluzione dello schermo e soprattutto il WiFi. Se non si vuole mantenere collegati al Raspberry Pi tastiera, monitor e mouse, conviene inoltre abilitare sulla scheda il protocollo SSH e la connessione da remoto tramite VNC.
Tutte le informazioni relative a queste configurazioni sono illustrate nella **documentazione ufficiale Raspberry Pi** [4]. Successivamente, aprire una finestra terminale e digitare i seguenti comandi per aggiornare la distribuzione all'ultima versione disponibile:

```
sudo atp-get update
sudo atp-get dist-upgrade
```

Una volta configurata la scheda, possiamo procedere con l'installazione delle librerie richieste dall'applicazione, in particolare la **libreria PyOTA** che espone le API per accedere al Tangle di IOTA tramite il linguaggio di programmazione Python.
La libreria PyIOTA, corredata delle istruzioni per l'installazione, è disponibile su GitHub [5]. Poiché installeremo la libreria PyIOTA sul Raspberry Pi, dovremo fare precedere i comandi di installazione indicati nelle istruzioni [5] dal **prefisso sudo**:

```
sudo pip install pyota
sudo pip install pyota[curl]
```

## IL CODICE PYTHON

Ora che il circuito è stato completamente assemblato e tutto il software di base con le relative librerie è stato installato sul nostro Raspberry Pi, possiamo cominciare ad esaminare il codice Python che avrà il compito di governare l'applicazione.
L'intero codice è racchiuso in un **unico script Python** (*let_there_be_light.py*), anche questo disponibile su GitHub [6]. Listato completo a pag. 123.

## ESECUZIONE E TEST

Per eseguire l'applicazione occorre anzitutto copiare lo script Python (dopo avere scaricato il relativo zip da GitHub) sul Raspberry Pi, oppure (meglio ancora) scaricarlo direttamente GitHub tramite git (anche lo strumento Git è già incluso nella distribuzione Raspbian).
Fatto ciò, è sufficiente aprire sul Raspberry Pi una finestra terminale, navigare fino alla cartella in cui è stato copiato lo script e digitare il seguente comando:

```
python let_there_be_light.py
```

Dovreste a questo punto vedere il codice in esecuzione nella finestra terminale, con la visualizzazione del saldo IOTA disponibile e il monitoraggio dei fondi disponibili **eseguito periodicamente ogni 10 secondi**.
Per accendere il LED è sufficiente utilizzare l'applicazione per wallet IOTA preferita e trasferire tramite essa una somma qualsiasi in IOTA con destinatario l'indirizzo associato al LED.
Non appena la transazione verrà confermata dal Tangle IOTA, il LED si accenderà e rimarrà acceso per un periodo funzione del valore di IOTA trasferito.
Nello script proposto, ad esempio, si è **simulato un**

**consumo di 1 IOTA per ogni secondo** di accensione del LED.
Se si vuole quindi che il LED rimanga acceso per un minuto, occorre fare una transazione di pagamento con inporto di 60 IOTA.
Se per il trasferimento si utilizza un'app, può essere conveniente stampare il QR code associato all'indirizzo IOTA su cui effettuare il trasferimento, in modo tale da velocizzare e semplificare l'operazione di pagamento stessa.

## CONCLUSIONI

Il progetto [7] presentato nell'articolo dimostra come un sistema embedded di tipo standard (nel nostro caso una comune scheda Raspberry Pi 3 B/B+) sia in grado di interfacciarsi e interagire con la rete IOTA.
Questa funzionalità, resa possibile dalla disponibilità di apposite librerie di interfacciamento scritte in Python, testimonia come IOTA e il Tangle abbiano tutti i requisiti per supportare diverse tipologie di applicazioni IoT.

## RIFERIMENTI


[1] https://thetangle.org
[2] https://www.raspberrypi.org/downloads/
[3] https://www.balena.io/etcher/
[4] https://www.raspberrypi.org/documentation/configuration/
[5] https://github.com/iotaledger/iota.lib.py
[6] https://gist.github.com/huggre/a3044e6094867fe04096e0c64dc60f3b
[7] https://medium.com/coinmonks/integrating-physical-devices-with-iota-83f4e00cc5bb


L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/come-integrare-dispositivi-iot-nella-rete-iota

# APPLICAZIONI DELLO STANDARD DDS AL MONDO IOT

di **Stefano Lovati**

*Dal punto di vista delle tecnologie e dei protocolli di comunicazione, il mondo di internet delle cose (IoT) si trova ancora in uno stadio iniziale di adozione. A differenza di altri contesti che comportano un elevato scambio di informazioni, la sfida che l'IoT deve affrontare è molto più ardua, legata soprattutto alla necessità di garantire prestazioni in tempo reale con elevato grado di sicurezza. DDS, acronimo di Data Distribution Service, è uno standard in grado di supportare i requisiti di performance e affidabilità imposti dalle applicazioni IoT.*

## INTRODUZIONE

**Data Distribution Service** (DDS) è un **middleware** concepito per i **sistemi distribuiti** basati sullo scambio di dati in tempo reale tra più sorgenti e più destinazioni. Lo standard **DDS** è perciò orientato alle applicazioni **M2M** (machine to machine) e **IoT**, dove agevola lo scambio dati grazie a un sistema di comunicazione con caratteristiche importanti quali: scalabilità, sicurezza, interoperabilità, efficienza e comportamento in tempo reale. Lo standard DDS, le cui specifiche iniziali furono emanate nel 2001 dal consorzio Object Management Group (OMG), è attualmente utilizzato in numerose applicazioni di tipo governativo, militare e industriale. Con riferimento alla **Figura 1** possiamo osser-

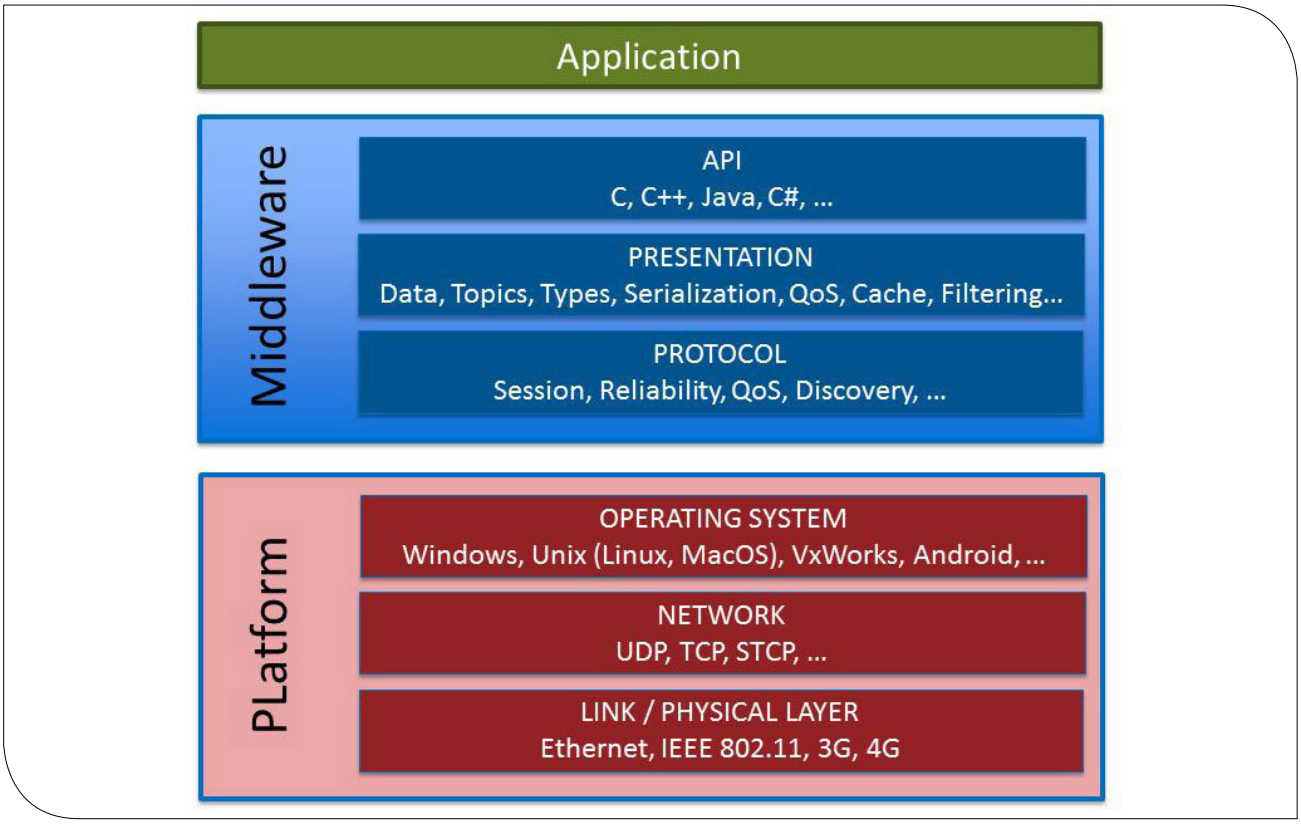



*Figura 1: il middleware DDS astrae l'applicazione dai dettagli di basso livello*

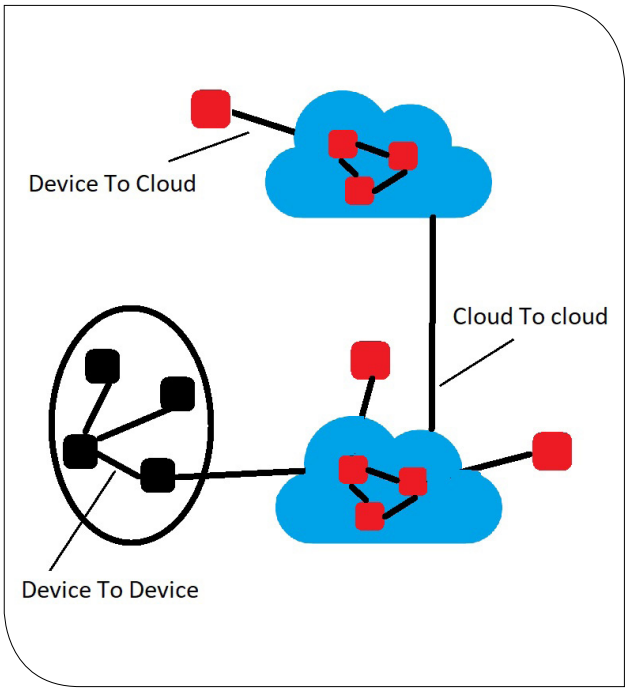



*Figura 2: le diverse tipologie di comunicazione tra dispositivi IoT e cloud*

vare come il middleware si collochi tra il livello dell'applicazione e quello del sistema operativo e consista in una serie di protocolli, librerie e API che definiscono un ben preciso framework.
Rispetto ad altri standard o protocolli di comunicazione, ciò che rende il **DDS unico** sono i seguenti **due aspetti**:

- è un sistema di scambio messaggi in grado di consentire la comunicazione tra uomo e macchina;
- è un sistema di messaggistica assimilabile a un "database mobile", nel senso che il protocollo è in grado di trasportare non solo dati ma anche intere tabelle di database, consentendo agli utenti di accedere alle informazioni in tempo reale tramite apposite chiavi e produrre report analitici e statistici.

Le **applicazioni** dello standard **DDS** in ambito **IoT** sono numerose e diversificate, si pensi ad esempio alle aziende che operano nel settore della distribuzione di energia o dello sfruttamento delle energie rinnovabili, al settore dei trasporti e della mobilità, sistemi di smart metering e smart building e altro ancora. Essendo un protocollo di comunicazione standard con caratteristiche real time, DDS è in grado di gestire efficacemente tutti questi tipi di applicazioni, con in più la possibilità di acquisire e trasmettere le informazioni da un nodo verso un altro o verso un server remoto. DDS è **scalabile** ed è perciò in grado di adattarsi a una rete composta da un numero elevato di nodi, acquisendo e veicolando le informazioni in una frazione di secondo.

## UBIQUITÀ DEI DATI

Il valore aggiunto (e in molti casi un requisito imprescindibile) di una qualunque applicazione IoT è la possibilità di disporre di informazioni puntuali ed aggiornate sul sistema ovunque ci si trovi nel mondo: è sufficiente disporre di una connessione ad internet. Questo concetto prende il nome di **ubiquità dei dati** e offre un potenziale elevatissimo in quanto aiuta le organizzazioni ad assumere le decisioni e a definire le strategie più corrette. Relativamente alle applicazioni IoT, possiamo definire tre diverse **tipologie di architetture** per i sistemi distribuiti, evidenziate nell'immagine di **Figura 2**:

- **Device To Device**: questo tipo di comunicazione viene utilizzato nelle applicazioni in cui i dispositivi devono condividere informazioni in modo efficiente, come impianti industriali, dispositivi mobile e nel settore automotive. L'efficienza è facilitata dalla mancanza di un broker all'interno della rete: ciascun dispositivo può colloquiare direttamente con gli altri (connessione peer to peer) senza dover passare da un nodo intermedio. Con questo tipo di comunicazione si riescono ad ottenere tempi di latenza ridotti ed elevata ampiezza di banda;
- **Device To Cloud**: in questo caso sia i singoli dispositivi sia intere parti della rete (sottosistemi) possono interagire con i servizi e le applicazioni fornite dal **cloud** al fine di scambiarsi delle informazioni o produrre dei report analitici. I requisiti relativi a questo tipo di comunicazione dipendono dal particolare contesto: un'applicazione di chirurgia remota avrà ad esempio dei requisiti temporali molto più stringenti rispetto a una normale applicazione di telelettura di uno smart meter. Oltre alle caratteristiche di bassa latenza ed elevato **throughput**, la comunicazione Device To Cloud deve essere in grado di operare su reti in cui possono sussistere restrizioni sulla banda o collegamenti funzionanti in modo intermittente;
- **Cloud To Cloud**: nonostante il numero di sistemi in grado di coprire geograficamente più stati o continenti sia ancora limitato, la tendenza attuale è quella di realizzare reti IoT che permettano, in modo semplice ed efficiente, di scambiare informazioni tra diversi domini cloud. A questo

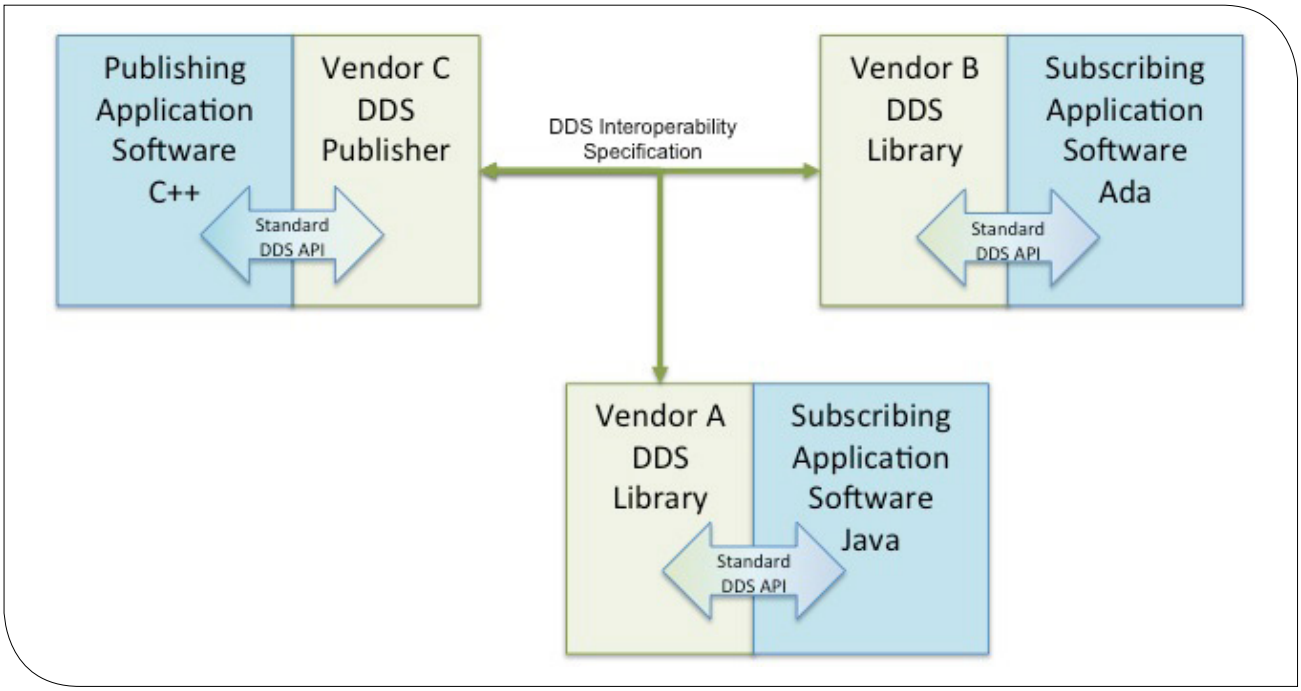



*Figura 3: l'interoperabilità permette l'integrazione tra diversi produttori*

livello, l'infrastruttura di comunicazione deve essere in grado di supportare l'istradamento "intelligente" delle informazioni, scegliendo volta per volta il percorso ottimale in grado di garantire il maggiore throughput e la minore latenza.

## IL DATA DISTRIBUTION SERVICE (DDS)

DDS è uno standard per lo scambio dati nei sistemi distribuiti che presenta non solo la caratteristica di ubiquità, ma anche efficienza, sicurezza, comportamento real time e indipendenza dalle piattaforme hardware e software utilizzate (si dice anche che è un sistema "cross platform"). DDS è sia un protocollo che permette l'**interoperabilità** tra dispositivi sviluppati da produttori diversi, sia un insieme di **API** (Application Programming Interface) che agevola il porting di un'applicazione su diverse piattaforme hardware/software. Lo standard impone che l'implementazione sia completamente distribuita e non richieda l'utilizzo di un concentratore (broker); si ottiene in questo modo una comunicazione senza la presenza di alcun mediatore e pertanto totalmente trasparente. In **Figura 3** possiamo osservare l'espressione del concetto di interoperabilità insito nel protocollo su cui si basa il DDS. Lo standard garantisce infatti che le applicazioni o le librerie sviluppate da produttori diversi possano integrarsi senza problemi rispettando il meccanismo della **Publish/Subscribe** di cui parleremo a breve.

Uno dei concetti che sta alla base del DDS è quello del **topic**, peraltro comune in altri protocolli utilizzati in ambito IoT come ad esempio l'**MQTT**. Il topic rappresenta l'informazione di base che viene scambiata tra i dispositivi e rappresenta l'astrazione di un oggetto che ha invece una valenza fisica. Esempi di topic possono essere la temperatura e l'umidità acquisiti da un sensore, la tensione e la corrente misurati da un multimetro e così via. Al topic è sempre associato un altro parametro fondamentale, la **QoS** (Quality of Service), che identifica il livello della qualità del servizio che si vuole ottenere (il massimo valore di QoS garantisce che non vi sia perdita di messaggi durante la comunicazione tra i nodi IoT). DDS mette a disposizione dell'utente un'ampia gamma di profili QoS, attraverso i quali è possibile controllare l'utilizzo delle risorse locali, l'utilizzo della rete, la differenziazione tra varie tipologie di traffico e l'immediata disponibilità dei dati (topic) ai nuovi nodi che si collegano alla rete. Nello standard DDS la generazione dei dati è effettuata dai nodi che rivestono il ruolo di **Data Writer** (concetto analogo a quello del **Publisher** di MQTT), mentre i nodi che "consumano" i dati vengono detti **Data Reader** (analoghi ai **Subscriber** di MQTT). Per ogni topic, ogni Data Reader può raffinare ulteriormente l'informazione ricevuta attraverso l'applicazione di opportuni **filtri temporali** e filtri sul **contenuto** dei dati ricevuti. Il protocollo dispone inoltre di un utile servizio dinamico di individuazione dei contenuti disponibili nella rete (detto **discovery**), consentendo all'applicazione di recuperare dinamicamente l'informazione richiesta individuando i

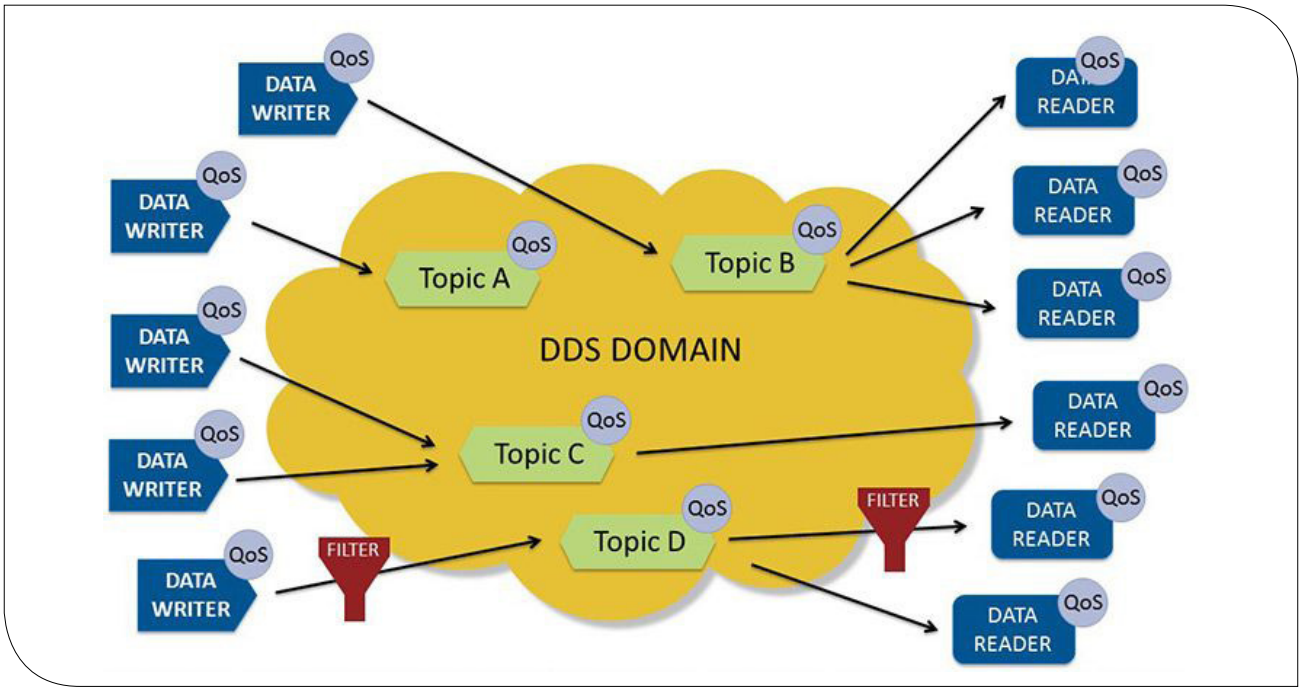



*Figura 4: DDS si basa sul pattern publish/subscribe*

nodi che la producono. Non da ultimo citiamo l'elevato grado di **sicurezza** offerto dal DDS, un framework estensibile in grado di gestire le procedure di autenticazione, cifratura e controllo degli accessi. In **Figura 4** è rappresentato un ipotetico dominio DDS in cui sono presenti più Data Writer (ognuno dei quali pubblica un topic con QoS associata) e più Data Reader.

## DDS E L'IOT

Tra gli standard considerati rilevanti per le applicazioni in ambito IoT, il DDS si contraddistingue soprattutto per la sua capacità di soddisfare i requisiti di scambio dei dati tra nodi e cloud. Con riferimento alle topologie di rete citate in precedenza, vediamo quali caratteristiche di DDS lo rendono così speciale rispetto ad altri standard:

- **Device To Device**: a questo livello DDS si dimostra una piattaforma molto **efficiente** e **scalabile**. La sua implementazione può infatti essere scalata verso il basso per soddisfare i requisiti delle applicazioni embedded, oppure verso l'alto per coprire le esigenze delle più avanzate applicazioni eseguite su macchine multi core. Oggi sono disponibili sul mercato implementazioni di DDS in grado di offrire una latenza inferiore a 30 µs su reti Gigabit Ethernet, con un traffico di diversi milioni di messaggi al secondo sui nodi terminali della rete. L'assenza di un broker (consentita dal pieno supporto dell'invio di messaggi multicast) e la sua natura insita di sistema peer to peer fanno del DDS la soluzione ideale per la comunicazione nelle reti Device To Device;
- **Device To Cloud**: a livello di comunicazione, DDS supporta diversi protocolli come **UDP** e **TCP/IP** e può trarre beneficio dall'utilizzo del multicast. Nonostante non sia in grado di garantire che ogni messaggio trasmesso sia stato correttamente consegnato al destinatario, l'UDP si dimostra particolarmente utile nelle applicazioni interattive, in cui non è richiesto un comportamento real time particolarmente spinto. In questi scenari il protocollo TCP/IP potrebbe infatti introdurre un sovraccarico eccessivo che inevitabilmente si riflette sulle prestazioni. La scelta di quando e come utilizzare il protocollo TCP/IP piuttosto che l'UDP dipende dalla qualità del servizio (QoS) desiderata. DDS prevede diversi **livelli di QoS**: dal **best effort** (il sistema farà il possibile affinché ogni topic pubblicato possa essere ricevuto dai nodi ad esso sottoscritti), passando per il **last value reliability**, fino ad arrivare al **reliability**. Quest'ultimo livello è quello che garantisce la massima affidabilità a livello di scambio dei messaggi ed è in questo senso paragonabile al protocollo TCP/IP. Viceversa, con gli altri livelli di QoS è ammissibile che alcuni dati "vecchi" vengano scartati per non ritardare

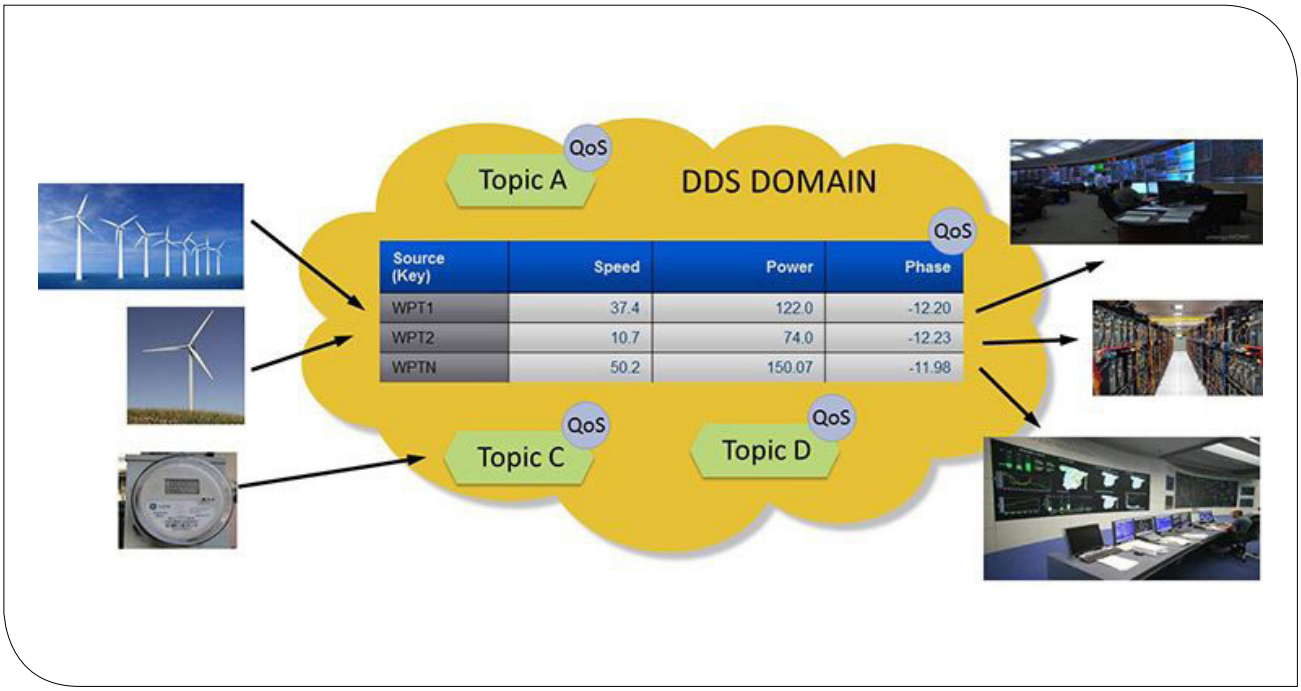



*Figura 5: la centralità dei dati nel DDS*

la trasmissione dei dati più "nuovi". DDS si dimostra così una soluzione eccellente per condividere sia **dati prodotti con periodicità** (si pensi alle applicazioni di telemetria) sia dati che richiedono un elevato grado di **affidabilità**. Inoltre, il supporto integrato per il filtraggio dei contenuti assicura che i dati sono inviati solo se esistono dei nodi consumatori che condividono gli stessi interessi e i cui filtri soddisfano i dati prodotti;

- **Cloud To Cloud**: anche in questo caso il DDS, in virtù della sua bassa latenza e dell'elevato throughput, si dimostra una soluzione ideale per supportare l'ingente mole di dati scambiata tra i sistemi cloud ospitati nei data center sparsi per il mondo.

## DISCOVERY

Discovery è una funzionalità insita nel protocollo DDS, eseguita automaticamente dal framework al verificarsi di determinati eventi. Più precisamente, questa operazione consiste nella creazione di una connessione tra un nodo subscriber e il nodo publisher che produce i dati (topic) a cui il subscriber stesso è interessato. Il processo ha inizio con l'invio, da parte del publisher, di un messaggio **multicast** con cui annuncia la disponibilità dei topic da esso pubblicati. I subscriber, a loro volta, rispondono in modalità **unicast** e viene così creata una connessione publisher-subscriber. Il processo Discovery non solo semplifica la fase di configurazione, ma consente anche di ottenere un'elevata scalabilità del sistema, che può così reagire in modo flessibile all'aggiunta o rimozione di nodi subscriber dalla rete.

## I FILTRI

In precedenza abbiamo detto che DDS permette di attivare appositi **filtri a livello di ogni publisher o subscriber**, in modo tale che i nodi possano ricevere solo i topic che soddisfano determinate condizioni. Il principale vantaggio che ne consegue è un utilizzo più efficiente della rete e della banda disponibili, con una conseguente riduzione della latenza. In DDS esistono fondamentalmente quattro tipi di filtro:

- **time-based**: è la capacità di sotto-campionare dei dati ad elevata frequenza, ottenendo un tasso di trasferimento degli stessi notevolmente inferiore;
- **liveliness**: consiste nell'associare un segnale "heartbit" a un publisher o a un subscriber, in modo tale che l'applicazione possa sapere se un determinato nodo è attivo e sta operando correttamente;
- **history**: consiste nella possibilità di memorizzare in RAM una serie di dati storici, sia sul lato publisher sia su quello subscriber;
- **lifespan**: consiste nell'associare un periodo di vita utile ad ogni dato; quando questo timeout viene superato, il dato viene automaticamente rimosso dalla rete;

- **ownership**: consente di associare più sorgenti ad uno stesso topic, essenzialmente per garantire una maggiore robustezza e tolleranza ai guasti. In tal caso occorre specificare quale sorgente ha priorità sulle altre (dette secondarie o di backup).

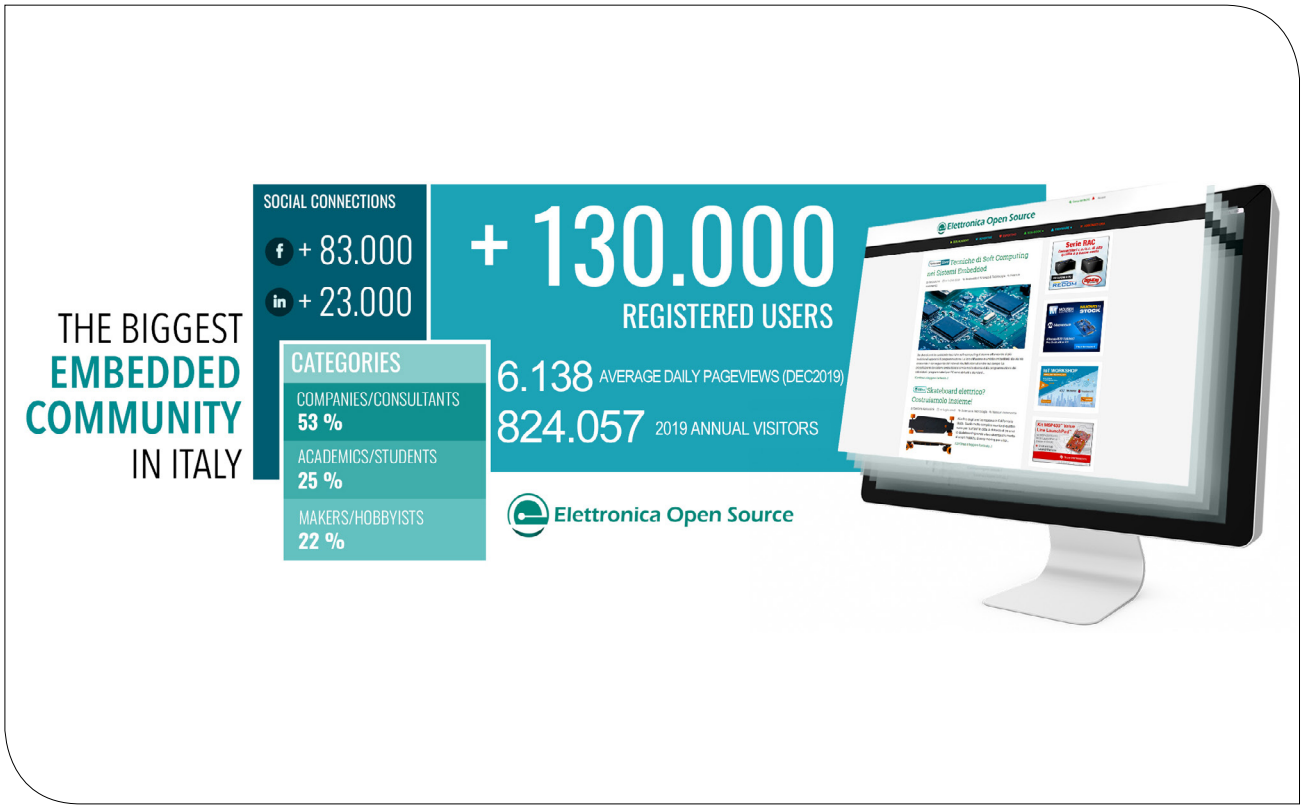
## CENTRALITÀ DEI DATI

La tecnica di comunicazione (o **pattern**) publish/subscribe è molto potente e nello stesso tempo intuitiva. Non è un caso se molti canali social media (tra cui Facebook e Twitter) usano proprio un sistema di questo tipo per gestire la pubblicazione dei post e l'invio delle corrispondenti notifiche agli utenti interessati.
Lo stesso pattern può essere applicato ai sistemi industriali, in cui le applicazioni si sottoscrivono alle variazioni delle misure provenienti dai sensori, alle notifiche di allarme, oppure ai comandi di sistema. La principale differenza esistente tra il DDS e protocolli analoghi per l'IoT basati su publish/subscribe (come l'MQTT) è la **centralità dei dati** che sta alla base del framework DDS.
Si osservi a tal proposito in **Figura 5** un esempio di applicazione DDS al centro della quale vi sono proprio i dati. Con il termine centralità dei dati si intende il fatto che i dati rappresentano niente di meno che l'interfaccia dell'applicazione, non vi è alcun bisogno di utilizzare dei contenitori artificiali come i messaggi e gli oggetti.

## CONCLUSIONI

Nel mondo IoT esistono diverse tecnologie in grado di trasmettere dati tra applicazioni diverse. Alcune sono basate sui protocolli UDP o TCP, altre si appoggiano a protocolli di alto livello come AMQP o MQTT. L'approccio comune seguito da questi sistemi si basa sull'invio di dati da un'applicazione a un'altra, ma **nessuna di queste è centralizzata sui dati**. L'applicazione che riceve i dati dovrà pertanto farsi carico di tutte le necessarie azioni di filtraggio e gestione dei dati.
Nel framework DDS, viceversa, tutte queste operazioni sono eseguite automaticamente e internamente al sistema stesso.
Filtraggio, QoS, discovery e pattern publish/subscribe sono tutte funzionalità che rendono il framework DDS adatto a gestire anche le configurazioni più complesse e critiche di rete IoT.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/applicazioni-dello-standard-dds-al-mondo-iot



# MICROCHIP SEMPLIFICA LA SICUREZZA IOT HARDWARE-BASED CON SOLUZIONI DI PRE-PROVISIONING

di **Andrea Garrapa**

*Con il proliferare del numero e dei tipi di dispositivi connessi, la frammentazione del mercato e le vulnerabilità in termini di sicurezza presenti nell'Internet of Things (IoT), si sono create sfide significative per gli sviluppatori. La sicurezza hardware-based è l'unico modo per proteggere le chiavi segrete dagli attacchi fisici e dall'estrazione remota, ma per configurare e predisporre ciascun dispositivo sono necessarie vaste competenze in sicurezza, oltre a tempi rilevanti di sviluppo e costi. Per rispondere a queste esigenza del mercato, soprattutto per distribuzioni di bassi volumi, Microchip Technology Inc. ha lanciato le prime soluzioni di pre-provisioning oggi disponibili per implementazioni di qualsiasi dimensione.*

## INTRODUZIONE

Oggi le aziende producono in tutto il mondo da centinaia a milioni di dispositivi connessi ogni anno, la scalabilità dell'architettura può rappresentare un grave ostacolo alle implementazioni. I produttori in genere sono stati in grado di supportare la configurazione e il provisioning solo per ordini di volume elevato, lasciando le aziende con implementazioni di dimensioni medio-piccole con opzioni di performance basse. **Microchip Technology Inc.** ha cercato di rispondere a queste mancanze grazie a delle soluzioni di pre-provisioning che forniscono l'archiviazione sicura delle chiavi grazie all'elemento di sicurezza **ATECC608A**. Queste soluzioni di pre-provisioning prendono il nome di **Trust Platform**. **Trust Platform** di Microchip per la sua famiglia di dispositivi *CryptoAuthentication* consente alle aziende di ogni dimensione di implementare facilmente l'autenticazione sicura. Tre elementi fondamentali contribuiscono al modello progettato per guidare le politiche per la sicurezza delle informazioni all'interno di un'organizzazione:

1. **Riservatezza**: i dati, memorizzati o in transito in un messaggio, devono essere visibili solo da persone autorizzate;
2. **Integrità**: un messaggio inviato non deve cambiare verso la sua destinazione; vuol dire garantire l'autenticità dell'informazione, ossia che tale informazione non venga alterata e che la sorgente dell'informazione sia autentica.
3. **Disponibilità**: significa che l'informazione è accessibile agli utenti autorizzati.

In questo contesto, la riservatezza è un insieme di regole che limita l'accesso alle informazioni, l'integrità è la garanzia che le informazioni siano affidabili e accurate e la disponibilità è una garanzia di accesso affidabile alle informazioni da parte di persone autorizzate. Tecnologie diverse contribuiscono a questi elementi, ma è comune tra loro l'uso di **chiavi segrete** o private che fanno parte di un tag di identificazione unico e verificabile. La modalità di gestione di tali chiavi, la loro memorizzazione e comunicazione, determina la sicurezza di un sistema. Il problema della sicurezza quindi sta tutto nella segretezza delle **chiavi**. Un sistema crittografico è sicuro nel momento in cui ogni cosa del sistema è di pubblico dominio tranne che per le **chiavi**. Le chiavi devono essere quindi ben nascoste e protette in un **elemento di sicurezza** che assicuri che esse non siano mai visibili.

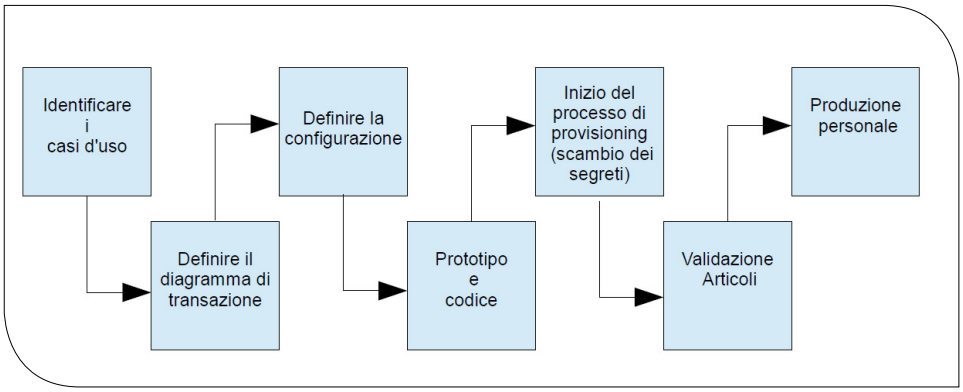



*Figura 1: schema a blocchi della personalizzazione di un elemento di sicurezza.*

Un **elemento di sicurezza** è in sostanza una cassaforte che protegge i segreti. Per un microcontrollore tale cassaforte è rappresentata da un dispositivo compagno (**ATECC608A**). I segreti sono invece le chiavi. Dentro a questa cassaforte i segreti (chiavi) vengono generati durante la fabbricazione dell'elemento, nelle fabbriche sicure di Microchip. I segreti (chiavi, certificati) non vengono mai pubblicamente esposti e vengono maneggiati grazie al servizio di provisioning sicuro di Microchip.

## ATECC608A

L'**ATECC608A** è un dispositivo di sicurezza progettato per supportare un'unità MCU host con una serie di sofisticate funzioni di sicurezza complementari attraverso una semplice interfaccia seriale. **ATECC608A** offre una archiviazione sicura delle chiavi, valutata "high" secondo i criteri comuni della *Joint Interpretation Library* (JIL), fornendo ai clienti la certezza che i dispositivi implementino pratiche di sicurezza comprovate nel settore ed il massimo livello di sicurezza nell'archiviazione delle chiavi. Con l'archiviazione *root of trust* basata su hardware e le contromisure crittografiche, il dispositivo protegge dalle classi più varie di attacchi fisici noti. Le strutture di produzione sicure di Microchip forniscono chiavi in modo sicuro, garantendo che le chiavi non vengano mai esposte a terzi durante il provisioning o la vita del dispositivo. I componenti della famiglia *CryptoAuthentication* di Microchip, di cui **ATECC608A** fa parte, rafforzano ulteriormente i meccanismi basati su hardware a protezione delle chiavi private e di altri dati che devono rimanere segreti. Il dispositivo vanta, fra le sue caratteristiche, l'accelerazione hardware per numerosi algoritmi, fra cui:

- Algoritmi di crittografia asimmetrica:
  - Algoritmo di firma digitale a curva ellittica (ECDSA) FIPS186-3
  - Diffie-Hellman a curva ellittica (ECDH) FIPS SP800-56A
  - Crittografia a curva ellittica (ECC) P256 standard NIST
  - Algoritmi di crittografia simmetrica:
  - Crittografia hash SHA-256
  - Crittografia con codice di autenticazione basato su hash (HMAC)
  - Crittografia AES-128
  - Crittografia AES-GCM (moltiplicazione del campo di Galois)
- Funzioni di derivazione della chiave (KDF):
  - Funzione pseudocasuale KDF (PRF)
  - KDF a estrazione ed espansione basata su HMAC (HKDF)

Queste funzionalità crittografiche rappresentano un elenco completo dei meccanismi richiesti per supportare protocolli di sicurezza di livello superiore per l'autenticazione e lo scambio sicuro di dati, come ad esempio per il **protocollo TLS** (*Transport Layer Security*). Il **protocollo di autenticazione TLS** è fondamentale per la sicurezza Internet. Per supportare questa componente fondamentale delle comunicazioni sicure è nata un'intera industria di provider di certificati, chiamati autorità certificative (CA). Le aziende acquisiscono dalle CA certificati attendibili da installare sui propri server a supporto del protocollo di autenticazione del server TLS standard.

## TRUST PLATFORM

La sfida nello sviluppo di un sistema di sicurezza sta nella personalizzazione. L'elemento di sicurezza necessità di essere "cucito" rispetto alle necessità del cliente, ma come? In figura 1 vengono riportati i passi da effettuare per raggiungere la personalizzazione dell'elemento di sicurezza.

Microchip ha lanciato la **Trust Platform** per la famiglia di dispositivi *CryptoAuthentication*. Si tratta di un servizio sicuro di provisioning per la famiglia di dispositivi di sicurezza di Microchip, con una suite di strumenti di sviluppo per tutti i volumi. **Trust Platform** permette:

- un più semplice onboarding grazie a predefiniti casi d'uso
- uno sviluppo rapido grazie a strumenti semplici
- architettura agnostica con qualsiasi tipo di cloud, PKI, controller o connettività

| Trust&GO | TrustFLEX | TrustCU-STOM | |
|---|---|---|---|
| Pre-configurato | si | si | si |
| Pre-provisioned | si | si (flessi-bile) | si |
| Minima quantità ordinabile (MOQ) | 10 unità | 2000 unità | 4000 unità |
| Tempo di svi-luppo | il più basso | basso | persona-lizzato |
| Complessità | il più basso | basso | persona-lizzato |
| Storage sicuro della chiave | JIL High | JIL High | JIL High |

*Tabella 1: principali differenza tra i 3 livelli **Trust Platform**.*

- sfruttamento semplificato dei flussi dei negozi di e-commerce
- adatto per il mercato di massa con basso MOQ, includendo servizio di provisioning e certificati di identificazione personale.

La **Trust Platform** di Microchip è composta da un'offerta su tre livelli, e fornisce elementi di sicurezza pronti all'uso, pre-configurati o completamente personalizzabili, consentendo agli sviluppatori di scegliere la piattaforma più adatta al loro specifico progetto.

## TRUST&GO

Come prima soluzione per fornire un'autenticazione sicura pronta all'uso per il mercato di massa, il primo livello – **Trust&GO** – fornisce elementi sicuri di pre-provisioning, zero touch, e con un MOQ (Minimum Orderable Quantity) di appena 10 unità. Le credenziali del dispositivo sono pre-programmate, inviate e bloccate all'interno dell'ATECC608A per autenticazione di onboarding automatico su cloud o LoRaWAN. Parallelamente, i certificati e le chiavi pubbliche corrispondenti vengono consegnati in un file "manifest", che può essere scaricato tramite lo store di e-commerce Microchip e alcuni partner di distribuzione selezionati. Oltre a risparmiare fino a diversi mesi sui tempi di sviluppo, la soluzione semplifica notevolmente la logistica di provisioning, rendendo facile per i clienti nel mercato di massa proteggere e gestire i dispositivi edge senza i costi generali dei servizi di provisioning di terzi o delle autorità di certificazione.

Possiamo riassumere le caratteristiche del primo livello **Trust&GO** nei seguenti punti:

- **facile da usare** - non è necessario comprendere a fondo concetti di sicurezza complessi e implementare l'installazione per i certificati, scegliere un certificato radice, definire un modello di autenticazione etc
- **pensato per distribuzioni molto basse (10 unità)** - fino ad ora il valore di MOQ più basso per ottenere supporto diretto era di 100k unità. Ma ora è possibile rendere disponibile l'archiviazione sicura delle chiavi per il mercato di massa
- **connettività/cloud indipendente** - per tutte le reti basate su protocollo TLS (AWS, Google, Microsoft, reti private) e supporto LoRa.
- **time-to-market ridotto** - maggiore autonomia, minore complessità (non c'è bisogno di coinvolgere esperti, tutto lo sviluppo avviene nella MCU con *CryptoAuthLib*), onboarding più semplice sfruttando i negozi di e-commerce Microchip e i partner di distribuzione.

Grazie alla possibilità di autenticarsi su qualsiasi infrastruttura cloud pubblica o privata, Trust Platform di Microchip è anche flessibile e personalizzabile. Per i clienti che desiderino una maggiore personalizzazione, il programma include le piattaforme **TrustFLEX** e **TrustCUSTOM**.

## TRUSTFLEX

Il secondo livello del programma, **TrustFLEX**, offre la flessibilità di utilizzare l'autorità di certificazione prescelta dal cliente pur beneficiando di casi d'uso preconfigurati. Questi casi d'uso includono misure di sicurezza di base come l'autenticazione rafforzata su Transport Layer Security (TLS) per la connessione a qualsiasi rete basata su IP utilizzando qualsiasi certificate chain, autenticazione LoRaWAN, secure boot, aggiornamenti Over-the-Air (OTA), protezione IP, protezione dei dati utente e rotazione delle chiavi. Ciò riduce i tempi e la complessità legati alla personalizzazione del dispositivo senza richiedere codici prodotto personalizzati.
Possiamo riassumere le caratteristiche del secondo livello **TrustFLEX** nei seguenti punti:

- **bassi MOQ (2k unità)** - soluzione di sicurezza senza compromessi per le masse, è in grado di affrontare tutti i tipi di progetti e dimensioni di volume dell'azienda, mentre il valore di MOQ dei concorrenti per ottenere supporto è di circa 100k unità.
- **la maggiore parte dei casi d'uso sono predefiniti** - supporto per qualsiasi servizio cloud, compatibile per supportare qualsiasi provider di autorità di certificazione ed accogliere i segreti del cliente
- **modello di coinvolgimento semplificato per il processo di provisioning** - Trust Platform Design Suite elimina gli attriti per lo sviluppo e lo scambio segreto.

## TRUSTCUSTOM

Per i clienti che desiderano personalizzare completamente i propri progetti, il terzo livello del programma, **TrustCUSTOM**, fornisce funzionalità di configurazione specifiche del cliente e provisioning personalizzato delle credenziali.
Possiamo riassumere le caratteristiche del secondo livello **TrustCUSTOM** nei seguenti punti:

- **bassi valori di MOQ (4k unità)** - personalizzazione accessibile per prodotti a basso volume
- **onboarding semplificato con nuovi strumenti** - sviluppo più semplice iniziale con strumento per casi d'uso, configurazione più semplice con strumento di configurazione, onboarding più semplice per scambi segreti con XML Generator
- **chiavi protette in key storage con livello High JIL**.

Nella tabella 1 vengono riassunte le differenze tra i tre livelli della **Trust Platform**. Microchip ha inoltre collaborato con *Amazon Web Services* (**AWS**) per consentire un processo di onboarding chiaro e semplificato nei servizi AWS IoT per prodotti progettati con tutte le varianti di Microchip Trust Platform.

## STRUMENTI DI SVILUPPO

ATECC608A può essere associato ad uno qualsiasi dei microcontrollori e microprocessori. Per la rapida prototipazione di soluzioni sicure i progettisti possono utilizzare la Trust Platform Design Suite, che include:

- Un "use case tool" guidato
- Guida passo passo eseguibile funzionante su notebook Jupyter
- Esempi di codice C per ognuno dei casi d'uso
- Una utility "secret exchange"
- Trust Platform hardware development kits

## CONCLUSIONE

L'aumento degli attacchi riusciti a soluzioni di sicurezza software-based sottolinea la necessità per le aziende di adottare le migliori pratiche possibili, incluso l'isolamento delle chiavi private in elementi di sicurezza (**ATECC608A**). **Trust Platform** di **Microchip** semplifica l'implementazione della sicurezza hardware-based per le aziende di ogni dimensione, eliminando le barriere tradizionalmente associate alla configurazione e al provisioning dei dispositivi. Grazie ad un'offerta di tre livelli, fornisce elementi di sicurezza pronti all'uso, pre-configurati o completamente personalizzabili, consentendo agli sviluppatori di scegliere la piattaforma più adatta al loro specifico progetto.

***Prezzi e disponibilità***

I dispositivi nella Trust Platform di Microchip sono disponibili in volumi di produzione con i seguenti MOQ (minimum order quantities):

- Trust&GO per TLS (ATECC608A-TNGTLSx-B): 1,20$ con un MOQ di 10 unità
- Trust&GO per TLS (ATECC608A-TNGTLSx-G): 0,77$ con un MOQ di 2000 unità
- Trust&GO per LoRaWAN (The Things Industries ATECC608A-TNGLORAx-B and Actility ATECC608A-TNGACTU-B): 1,40$ con un MOQ di 10 unità
- TrustFLEX per LoRaWAN any join servers (ATECC608A-TFLXLORAx): 0,938$ con un MOQ di 2000 unità
- TrustFLEX (ATECC608A-TFLXTLSx): 0,845$ con un MOQ di 2000 unità
- TrustCUSTOM (ATECC608A-TCSTMTLSx): 0,883$ con un MOQ di 4000 unità

Gli strumenti di sviluppo nella Trust Platform di Microchip sono disponibili a:

- CryptoAuth Trust Platform kit: 13$
- ATECC608a Trust Platform kit: 14$

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/microchip-semplifica-la-sicurezza-iot-hardware-based-con-soluzioni-di-pre-provisioning

# DOMUS 1.0 – SISTEMA DI CONTROLLO CON GESTIONE REMOTA VIA WEB

di **Roberto Baresi**

*Sistema basato su Arduino UNO WiFi per il controllo locale e remoto fino ad un massimo di 5 apparecchiature. (luci, riscaldamento, ecc.) con rilevamento di temperatura e umidità in tempo reale fino ad un massimo di due ambienti. Le apparecchiature possono essere attivate sia localmente, con indicazione visiva dello stato tramite LED, sia da remoto con una semplice interfaccia WEB implementata direttamente sulla scheda Arduino.*

## L'ESIGENZA

L'esigenza nasce dalla necessità di controllare i sistemi di riscaldamento di una Scuola di Danza di 400 mq in provincia di Brescia. Questo sistema ha permesso l'ottimizzazione dei tempi di attivazione dei 3 sistemi di riscaldamento a gas della scuola, con un notevole abbattimento dei consumi e conseguente risparmio sulle bollette.

L'edificio, composto da due sale da ballo e un'area spogliatoi e servizi, è un capannone industriale adattato allo

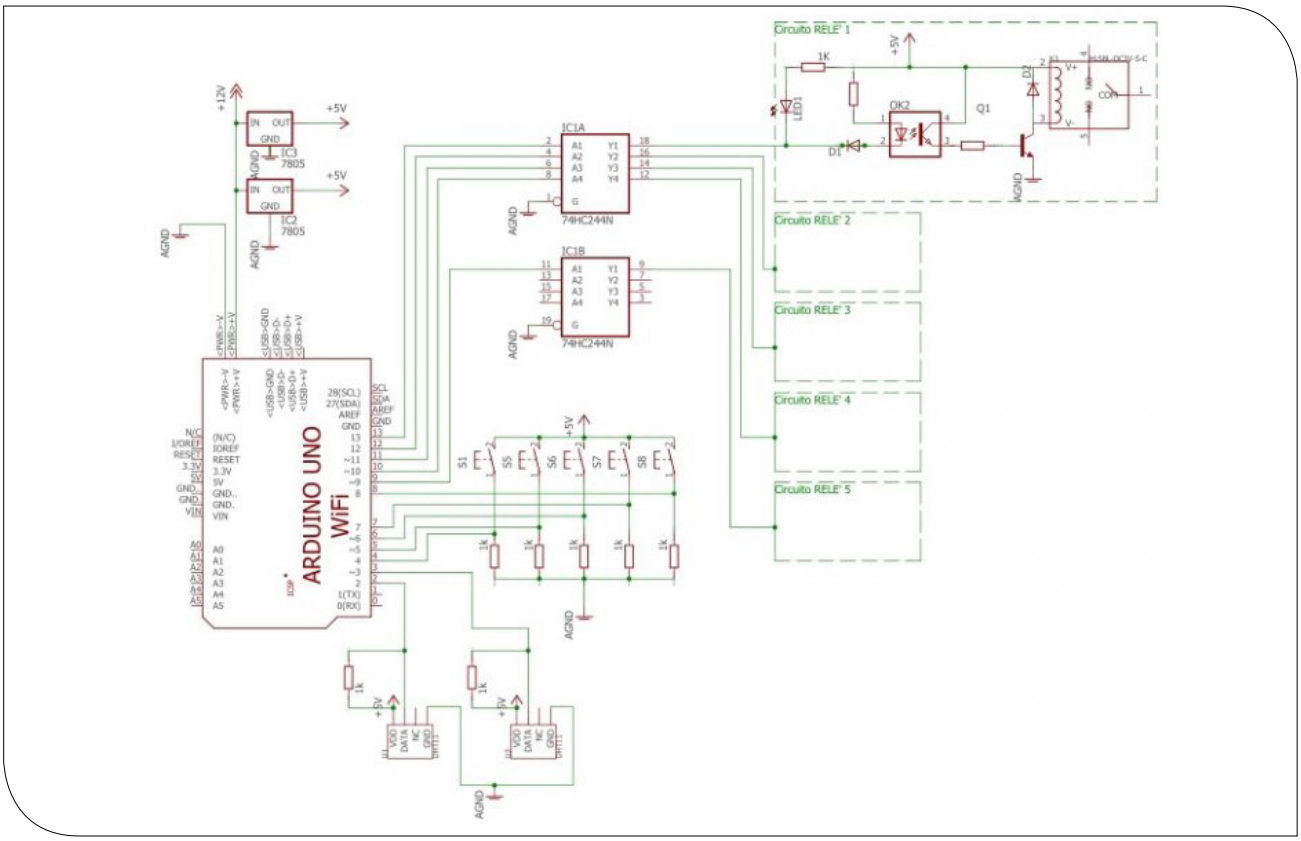



*Figura 1: Schema Elettrico*

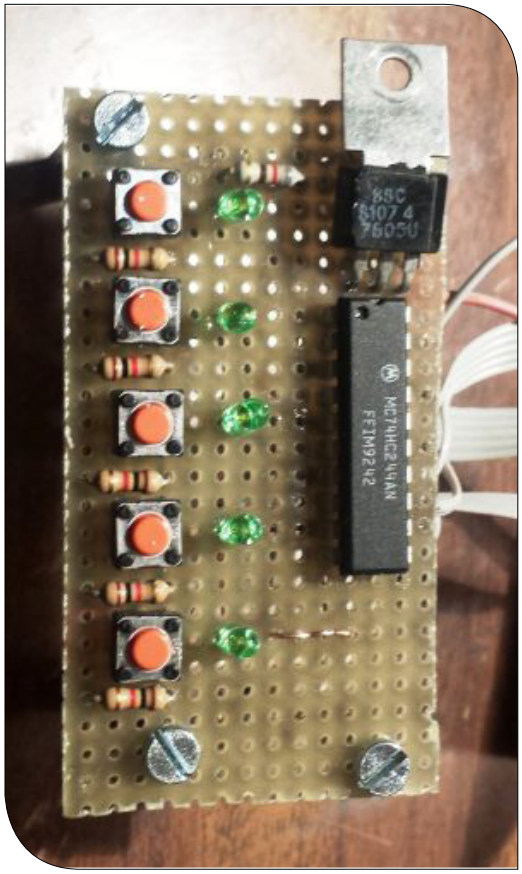


*Figura 2: particolare del pannello pulsanti e LED*

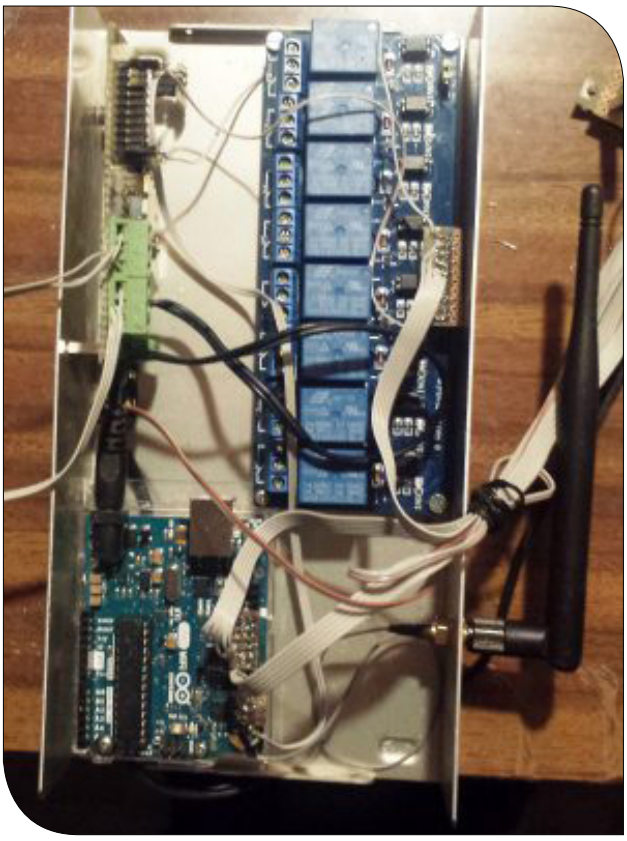


*Figura 3: vista d'insieme del sistema*

scopo ma che purtroppo non offre garanzie in termini di isolamento termico, i consumi di gas per il riscaldamento (e le conseguenti bollette) sono quindi di notevole entità e richiedono una particolare attenzione ai momenti di accensione e spegnimento dei 3 sistemi di riscaldamento separati per evitare di riscaldare inutilmente la scuola quando non ci sono lezioni ma di poterli accendere con un ragionevole anticipo rispetto all'orario delle lezioni per evitare di far trovare gli ambienti freddi.

I responsabili della scuola abitano a 10km di distanza dall'edificio e quindi si è reso necessario realizzare un sistema di controllo via rete (la scuola è dotata di connessione adsl con IP statico) che permettesse ai gestori di accendere e spegnere da remoto via rete i sistemi e controllarne lo stato, come pure controllare la temperatura e l'umidità delle due sale.

All'interno della scuola operano anche altre persone esterne all'associazione, era quindi necessario implementare un sistema con un doppio controllo: sia da remoto via web, sia in locale con comandi manuali.

Le due modalità dovevano inoltre essere attive contemporaneamente e sincronizzate tra di loro: si doveva poter accendere un riscaldamento da locale e spegnerlo da remoto e viceversa e soprattutto controllare da remoto che non fossero stati dimenticati accesi i sistemi di riscaldamento a scuola chiusa.

## LA SOLUZIONE IMPLEMENTATA

Il progetto è stato realizzato intorno alla **scheda Arduino Uno WiFi** che è una Arduino Uno con Wifi integrato, basata sul processore Atmega328P e il modulo WiFi ESP8266.

Il modulo WiFi ESP8266 è un microcontrollore da 80MHz che fornisce accesso ad una rete WiFi e dispone di front-end WiFi, con funzioni cioè sia di client che di access point, e di uno stack TCP/IP con supporto DNS.

Un'utile funzione della Arduino Uno WiFi è il supporto per la programmazione OTA, sia per il trasferimento degli sketch Arduino che per il firmware WiFi.

Questo permette di riprogrammare l'intero sistema, ad esempio per fare upgrade di nuove versioni software, senza la necessità di collegare fisicamente il cavo USB alla scheda stessa.

Questa funzionalità si rivela particolarmente utile in applicazioni tipo quella che stiamo descrivendo, dove non sempre è agevole raggiungere il dispositivo installato per connettere direttamente il PC da cui scaricare il software.

La scheda Arduino UNO è dotata di 6 input analogici, un quarzo a 16MHz, un connettore USB, un jack per

```
[code]
 /*************************************
 * *********** DOMUS 1.0 ***************
 * ****** 2017 Roberto Baresi **********
 * *************************************

on your browser, you type http://<IP>/arduino/webserver/

Possible commands to send from the xmpp client:
 "/arduino/digital/PIN" -> to read a digital PIN
 "/arduino/digital/PIN/VALUE"-> to write a digital PIN (VALUE:1/0)

Example:
 "/arduino/webserver"-> Display Temp, Igro and Status of switches
 "/arduino/digital/13/1" -> digitalWrite(13, ON)
 "/arduino/digital/13/0" -> digitalWrite(13, OFF)
 */

#include <Wire.h>
#include <ArduinoWiFi.h>
#include <dht.h>

dht DHT;

#define Sensore1 3
#define Sensore2 2

String Sw[5] = {"9 Non attivo","10 Non attivo","11 Sala 2","12 Caldaia","13 Sala1"};

void setup()
    {
    pinMode(0, OUTPUT);
    pinMode(1, OUTPUT);
    pinMode(2, INPUT);
    pinMode(3, INPUT);
    pinMode(4, INPUT);
    pinMode(5, INPUT);
    pinMode(6, INPUT);
    pinMode(7, INPUT);
    pinMode(8, INPUT);
    pinMode(9, OUTPUT);
    pinMode(10, OUTPUT);
    pinMode(11, OUTPUT);
    pinMode(12, OUTPUT);
    pinMode(13, OUTPUT);

    digitalWrite(0, HIGH);
    digitalWrite(1, HIGH);
    digitalWrite(9, HIGH);
    digitalWrite(10, HIGH);
    digitalWrite(11, HIGH);
    digitalWrite(12, HIGH);
     digitalWrite(13, HIGH);
```

```
    Wifi.begin();
    Wifi.println("Domus 1.0 Roby Baresi RobyBaresi@gmail.com");
    String command = Wifi.readStringUntil('/');
    }

void loop()
  {
  // routine di gestione pulsanti
  for (int index = 4; index <= 8; index++)
    {
    int Pulsante = digitalRead(index);
    int Contatto = digitalRead(index + 5);
    if (Pulsante == HIGH) digitalWrite(index + 5, !Contatto);
    delay (50);
    }

  while (Wifi.available())
    {
    // read the command
    String command = Wifi.readStringUntil('/');
    if (command == "digital") Comanda();
    if (command == "webserver") Visualizza();
    }
  }

void Comanda()
  {
  int pin, value;
  pin = Wifi.parseInt();
  if (Wifi.read() == '/')
    {
    value = Wifi.parseInt();
    if (value==1) value=0;
    else value=1;
    digitalWrite(pin, value);
    }
  // scrivo pagina HTML
  Wifi.println("HTTP/1.1 200 OK");
  Wifi.println("Content-Type: text/html");
  Wifi.println("Refresh: 5"); // refresh the page automatically every 5 sec
  Wifi.println();
  Wifi.println("<!DOCTYPE html>");
  Wifi.print("<html>");
  Wifi.print("GB StudioDanza ASD<br><br>");
  Wifi.print("/<html>");
  Wifi.println();
  Wifi.print(DELIMITER); // very important to end the communication !!!
  }

void Visualizza()
  {
   // lettura temperature e umiditÃ
```

```
int chk = DHT.read11(Sensore1);
long T1 = DHT.temperature;
long H1 = DHT.humidity;
chk = DHT.read11(Sensore2);
long T2 = DHT.temperature;
long H2 = DHT.humidity;

// scrivo pagina HTML
Wifi.println("HTTP/1.1 200 OK");
Wifi.println("Content-Type: text/html");
Wifi.println("Refresh: 5"); // refresh the page automatically every 5 sec
Wifi.println();
Wifi.println("<!DOCTYPE html>");
Wifi.print("<html>");
Wifi.print("GB StudioDanza ASD<br><br>");

// temperatura e umidità
Wifi.print("Sala1 Temp --> ");
Wifi.print(T1);
Wifi.println("<br>");
Wifi.print("Sala1 Igro --> ");
Wifi.print(H1);
Wifi.println("%<br>");
Wifi.print("Sala2 Temp -->");
Wifi.print(T2);
Wifi.println("<br>");
Wifi.print("Sala2 Igro -->");
Wifi.print(H2);
Wifi.println("%<br><br>");

// porte digitali

for (int DigitalChannel = 9; DigitalChannel <= 13; DigitalChannel++)
  {
  String STATO;
  int sensorReading = digitalRead(DigitalChannel);
  if (sensorReading == LOW) STATO = " ACCESO";
  else STATO = " SPENTO";
  Wifi.print(Sw[DigitalChannel - 9]);
  // Wifi.print(DigitalChannel);
  Wifi.print(STATO);
  Wifi.println("<br>");
  }

// chiusura pagina html
Wifi.println("</html>");
Wifi.println();
Wifi.print(DELIMITER); // very important to end the communication !!!
}

[/code]
```

l'alimentazione, un connettore per la programmazione ICSP, un pulsantino per il reset della scheda e 14 pin di input/output digitali (6 dei quali possono essere usati come segnali PWM).
Ognuno dei 14 pin digitali di input/output della scheda Arduino Uno WiFi opera con una tensione di 5V e può essere usato sia come ingresso che come uscita, utilizzando le funzioni pinMode(), digitalWrite() e digitalRead().
Ogni pin può fornire o ricevere un massimo di 40mA e dispone di una resistenza di pull-up interna (disconnessa di default) di 20-50 Kohm.
Il circuito che pilota i relè è del tipo a "stato basso", cioè il relè si eccita e chiude il contatto in presenza di uno "ZERO" logico al piedino corrispondente di ingresso.
Un'importante caratteristica di questo modulo è la possibilità di distinguere l'alimentazione fornita ai relè dall'alimentazione fornita alla logica della scheda. Sul modulo è infatti presente un jumper a 3pin (GND-VCC-JDVCC) che permette di selezionare la fonte di alimentazione per i relè.
Quando il ponticello è posizionato sui pin VCC/JD-VCC, la tensione fornita per alimentare la logica tramite il connettore a 10 pin viene utilizzata anche per alimentare i relè.
Se invece si volesse utilizzare una fonte di alimentazione differente per pilotare i relè, è sufficiente scollegare il ponticello e collegare la fonte di alimentazione esterna direttamente tramite i pin GND e JD-VCC del jumper (lasciando scollegato il pin VCC centrale del ponticello).
In questo modo, l'alimentazione per la logica del modulo viene fornita tramite il connettore a 10 pin posto sulla scheda, mentre l'alimentazione per i relè viene prelevata dal jumper.
Potete acquistare questo banco relè, chiaramente visibile in Figura 3, sia su Amazon sia qui: https://www.robotstore.it/product/682/Modulo-Rel%C3%A8-8-canali-DC-5V-per-Arduino-e-Raspberry-Pi.html
Sono disponibili varie versioni che differiscono solo per il numero di relè presenti sulla scheda.
Anche se nel progetto in questione è stato usato il modello a 8 relè, il sistema ne pilota solo i primi cinque, non avendo la Arduino Uno WiFi un numero sufficiente di porte per pilotarli tutti e 8 e contemporaneamente gestire 8 pulsanti per l'attivazione/disattivazione.
Per ottenere la corrente necessaria per pilotare le logiche di ingresso dei Relè e contemporaneamente i LED posizionati sul pannello switch montato all'esterno del contenitore e visibili dall'esterno, le uscite della scheda Arduino (dalla 9 alla 13) si interfacciano con un **integrato 74LS244**. In Figura 2 è visibile l'integrato montato accanto agli switches.
Il 74LS240, (con le sue varianti 241 e 244) è un buffer ottale e driver di linea, con uscite a 3 stati programmabili, progettato per essere impiegato come driver per l'indirizzamento di memorie, come driver di clock o come driver per trasmettitori e ricevitori di dati su bus, le sue caratteristiche più importanti sono:

- Isteresi agli ingressi per migliorare i margini di rumore
- Uscite a 3 stati per pilotare linee BUS o circuiti di indirizzamento di memorie a stato solido.
- Il diodo clamp all'ingresso limita gli effetti di cambi di stato troppo veloci.

Questo integrato, reperibile a basso costo presso qualsiasi rivenditore di componenti elettronici, è in grado di pilotare senza alcun problema i LED di controllo visivo dello stato delle porte e contemporaneamente gestire gli stati logici all'ingresso dei circuiti dei relè.
Come rilevatori di temperatura ed umidità sono stati usati dei classici (e molto economici) sensori **DHT11**, montati su basette separate su cui è stata montata anche la resistenza di polarizzazione in modo da avere solo 3 fili tra il sensore e l'unità centrale.
**Il sensore DHT11** è un sensore di temperatura e umidità con uscita dei dati in formato digitale.
Il sensore utilizza una tecnica digitale esclusiva che unita alla tecnologia di rilevamento dell'umidità, ne garantisce l'affidabilità e la stabilità.
I suoi elementi sensibili sono connessi con un processore 8-bit single-chip.
Ogni sensore di questo modello è compensato in temperatura e calibrato in un'apposita camera di calibrazione che determina in modo preciso il valore di calibrazione il cui coefficiente viene salvato all'interno della memoria OTP.
Le sue piccole dimensioni e suo basso consumo unite alla lunga distanza di trasmissione (20 m) permettono al sensore DHT11 di essere adatto per molti tipi di applicazioni.
Il package con quattro pin in linea ne rendono facile la connessione.
Caratteristiche principali del sensore sono:

- Rilevazione contemporanea di umidità relativa e temperatura
- Segnale digitale calibrato
- Eccezionale stabilità a lungo termine
- Componenti aggiuntivi non necessari
- Distanza di trasmissione lunga (20m)

- Basso assorbimento
- Contenitore a 4 pin
- Alimentazione: 3-5.5V DC
- Segnale di uscita digitale del segnale tramite single-bus
- Elemento sensibile: Resistenza in Polimero
- Campo di misura umidità: 20-90% di umidità relativa
- Campo di misura temperatura di 0-50 gradi Celsius
- Precisione: umidità + -4% RH (Max + -5% di umidità relativa)
- Precisione temperatura +-2.0Celsius
- Risoluzione umidità: 1% di umidità relativa
- Risoluzione temperatura: 0.1Celsius
- Isteresi Umidità: + -1% RH
- Stabilità a lungo termine: + -0.5% UR / anno
- Tempo di rilevazione medio: 2s

Non avendo necessità di grossa precisione, è stato scelto questo sensore per la sua economicità, nulla vieta ovviamente di sostituirlo con sensori più precisi e con risoluzione maggiore.
Sul mercato ne esistono anche con rilevazione della pressione atmosferica, adatti a realizzare anche una vera e propria stazione meteorologica.
L'alimentazione è esterna tramite un classico alimentatore switch a 12V.
All'interno dell'unità, due classici regolatori di tensione **7805** ricavano dalla tensione principale a 12V, l'uno i 5V necessari per pilotare il banco relè, l'altro per alimentare il 74LS244 e i LED esterni.
Dovendo avere il doppio comando da remoto e locale, sono stati montati pulsanti sempre aperti , normalmente polarizzati a zero e quindi che tengono l'ingresso di Arduino (ingressi dal 4 all'8) normalmente in stato basso. La pressione del pulsante porta l'ingresso a stato ALTO e questo viene rilevato dal software che inverte lo stato della corrispondente porta di uscita.
Facendo un esempio: premendo il pulsante all'ingresso 4, il software sente il cambio di stato da ZERO a UNO, legge lo stato della porta 9 e lo inverte.
Le porte in uscita sono contemporaneamente gestite dalla routine software che si attiva quando nel browser digitiamo il classico comando "<IP Address>/arduino/digital/9/x" dove x sta per 1 (accende) o 0 (spegne)
Sia che si prema il pulsante sia che si dia il comando via browser, il LED corrispondente viene pilotato segnalando sempre e comunque in locale lo stato della porta, lo stesso avviene via browser quando si leggono gli stati e le temperature/umidità delle sale con il comando "<IP Address>/arduino/webserver".

## SCHEMA ELETTRICO E LISTA COMPONENTI

### LISTA COMPONENTI

- 1 x Arduino UNO WiFi
- 1 x 74ls244
- 1 x tinxi® 8 canali Relè Modulo 5V DC 230V per Arduino (acquistabile su Amazon)
- 10 x Resistenza 1k
- 4 x pulsanti (normalmente aperti)
- 2 x 7805 (regolatori di tensione
- 1 x alimentatore switch 12v (2A)
- 1 x Antenna WiFi (io l'ho recuperata da un vecchio router ADSL guasto)
- 4 x LED

## SOFTWARE

Il software è abbastanza rozzo e basilare, questo è dovuto al fatto che la Arduino Uno Wifi non ha molta memoria per le applicazioni e qualsiasi tentativo fatto di implementare un'interfaccia web più sofisticata, con bottoni ed un minimo di grafica, è fallito per problemi di memoria.
Anche l'utilizzo di un SD è risultato impossibile in quanto il caricamento della libreria necessaria alla gestione del modulo SD più le altre librerie saturavano la memoria immediatamente.
Di seguito lo sketch di gestione:

## FUTURE IMPLEMENTAZIONI

Oltre allo sviluppo dell'interfaccia WEB, è in programma lo sviluppo di una app per Android per pilotare il sistema da smartphone o tablet.
Già oggi è comunque possibile gestire da remoto il sistema via smartphone utilizzando il browser all'interno del dispositivo: l'interfaccia è scritta in HTML base e quindi funziona senza problemi su qualsiasi browser, anche il più primitivo.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo. Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
https://it.emcelettronica.com/domus-1-0-sistema-di-controllo-con-gestione-remota-via-web

# + 130.000
## REGISTERED USERS

**6.138** AVERAGE DAILY PAGEVIEWS (DEC2019)

**824.057** 2019 ANNUAL VISITORS

# THE BIGGEST **EMBEDDED COMMUNITY** IN ITALY

## SOCIAL CONNECTIONS

f + 83.000

in + 23.000

## CATEGORIES

COMPANIES/CONSULTANTS
**53 %**

ACADEMICS/STUDENTS
**25 %**

MAKERS/HOBBYISTS
**22 %**

Elettronica Open Source